

Article

Self-Adjusting Variable Neighborhood Search Algorithm for Near-Optimal k-Means Clustering

Lev Kazakovtsev *, Ivan Rozhnov , Aleksey Popov and Elena Tovbis 

Reshetnev Siberian State University of Science and Technology, Institute of Informatics and Telecommunications, Krasnoyarskiy Rabochiy av. 31, 660037 Krasnoyarsk, Russia; ris2005@mail.ru (I.R.); vm_popov@sibsau.ru (A.P.); sibstu2006@rambler.ru (E.T.)

* Correspondence: levklevk@gmail.com

Received: 9 October 2020; Accepted: 2 November 2020; Published: 5 November 2020



Abstract: The k-means problem is one of the most popular models in cluster analysis that minimizes the sum of the squared distances from clustered objects to the sought cluster centers (centroids). The simplicity of its algorithmic implementation encourages researchers to apply it in a variety of engineering and scientific branches. Nevertheless, the problem is proven to be NP-hard which makes exact algorithms inapplicable for large scale problems, and the simplest and most popular algorithms result in very poor values of the squared distances sum. If a problem must be solved within a limited time with the maximum accuracy, which would be difficult to improve using known methods without increasing computational costs, the variable neighborhood search (VNS) algorithms, which search in randomized neighborhoods formed by the application of greedy agglomerative procedures, are competitive. In this article, we investigate the influence of the most important parameter of such neighborhoods on the computational efficiency and propose a new VNS-based algorithm (solver), implemented on the graphics processing unit (GPU), which adjusts this parameter. Benchmarking on data sets composed of up to millions of objects demonstrates the advantage of the new algorithm in comparison with known local search algorithms, within a fixed time, allowing for online computation.

Keywords: cluster analysis; k-means; variable neighborhood search; agglomerative clustering; GPU

1. Introduction

1.1. Problem Statement

The aim of a clustering problem solving is to divide a given set (sample) of objects (data vectors) into disjoint subsets, called clusters, so that each cluster consists of similar objects, and the objects of different clusters have significant dissimilarities [1,2]. The clustering problem belongs to a wide class of unsupervised machine learning problems. Clustering models involve various similarity or dissimilarity measures. The k-means model with the squared Euclidean distance as a dissimilarity measure is based exclusively on the maximum similarity (minimum sum of squared distances) among objects within clusters.

Clustering methods can be divided into two main categories: hierarchical and partitioning [1,3]. Partitioning clustering, such as k-means, aims at optimizing the clustering result in accordance with a pre-defined objective function [3].

The k-means problem [4,5], also known as minimum sum-of-squares clustering (MSSC), assumes that the objects being clustered are described by numerical features. Each object is represented by a point in the feature space $\mathbb{R}^d$ (data vector). It is required to find a given number k of cluster centers

(called centroids), such as to minimize the sum of the squared distances from the data vectors to the nearest centroid.

Let $A_1, \dots, A_N \in \mathbb{R}^d$ be data vectors, N be the number of them, and $S = \{X_1, \dots, X_k\} \subset \mathbb{R}^d$ be the set of sought centroids. The objective function (sum of squared errors, SSE) of the k-means optimization problem formulated by MacQueen [5] is:

$$SSE(X_1, \dots, X_k) = SSE(S) = \sum_{i=1}^N \min_{X \in \{X_1, \dots, X_k\}} \|A_i - X\|^2 \rightarrow \min_{X_1, \dots, X_k \in \mathbb{R}^d}. \quad (1)$$

Here, $\|\cdot\|$ is the Euclidean distance, integer k must be known in advance.

A cluster in the k-means problem is a subset of data vectors for which the specified centroid is the nearest one:

$$C_j = \{A_i, i = \overline{1, N} \mid \|A_i - X_j\| = \min_{X \in \{X_1, \dots, X_k\}} \|A_i - X\|\}, \quad j = \overline{1, k}.$$

We assume that a data vector cannot belong to two clusters at the same time. At an equal distance for several centroids, the question of assignment to a cluster can be solved by clustering algorithms in different ways. For example, a data vector belongs to a cluster lower in number:

$$C_j = \left\{ A_i, i = \overline{1, N} \mid \nexists j' = \overline{1, k} : \|A_i - X_{j'}\| < \|A_i - X_j\| \text{ or } (\|A_i - X_{j'}\| = \|A_i - X_j\| \text{ and } j' > j) \right\}, \quad j = \overline{1, k}. \quad (2)$$

Usually, for practical problems with sufficiently accurate measured values of data vectors, the assignment to a specific cluster is not very important.

The objective function may also be formulated as follows:

$$SSE(X_1, \dots, X_k) = \sum_{j=1}^k \sum_{i=\overline{1, N}: A_i \in C_j} \|A_i - X_j\|^2 \rightarrow \min_{X_1, \dots, X_k \in \mathbb{R}^d}, \quad (3)$$

or

$$SSE(C_1, \dots, C_k) = \sum_{j=1}^k \sum_{i=\overline{1, N}: A_i \in C_j} \|A_i - X_j\|^2 \rightarrow \min_{C_1, \dots, C_k \subset \{A_1, \dots, A_N\}}. \quad (4)$$

Equations (3) and (4) correspond to continuous and discrete statements of our problem, respectively.

Such clustering problem statements have a number of drawbacks. In particular, the number of clusters k must be given in advance, which is hardly possible for the majority of practically important problems. Furthermore, the adequacy of the result in the case of a complex cluster shapes is questionable (this model is proved to work fine with the ball-shaped clusters [6]). The result is sensitive to the outliers (standalone objects) [7,8] and depends on the chosen distance measure and the data normalization method. This model does not take into account the dissimilarity between the objects in different clusters, and the application of the k-means model results in some solution $X_1, \dots, X_k$ even in the cases with no cluster structure in the data [9,10]. Moreover, the NP-hardness [11,12] of the problem makes the exact methods [6] applicable only for very small problems.

Nevertheless, the simplicity of the most commonly used algorithmic realization as well as the interpretability of the results make the k-means problem the most popular clustering model. Developers' efforts are focused on the design of heuristic algorithms that provide acceptable and attainable values of the objective function.

1.2. State of the Art

The most commonly used algorithm for solving problem (1) is the Lloyd's procedure proposed in 1957 and published in 1982 [4], also known as the k-means algorithm, or alternate location-allocation (ALA) algorithm [13,14]. This algorithm consists of two simple alternating steps, the first of which solves the simplest continuous (quadratic) optimization problem (3), finding the optimal positions of the centroids $X_1, \dots, X_k$ for a fixed composition of clusters. The second step solves the simplest combinatorial optimization problem (4) by redistributing data vectors between clusters at fixed positions of the centroids. Both steps aim at minimizing the SSE. Despite the theoretical estimation of the computational complexity being quite high [15–17], in practice, the algorithm quickly converges to a local minimum. The algorithm starts with some initial solution $S = \{X_1, \dots, X_k\}$, for instance, chosen at random, and its result is highly dependent on this choice. In the case of large-scale problems, this simple algorithm is incapable of obtaining the most accurate solutions.

Various clustering models are widely used in many engineering applications [18,19], such as energy loss detection [20], image segmentation [21], production planning [22], classification of products such as semiconductor devices [23], recognition of turbulent flow patterns [24], and cyclical disturbance detection in supply networks [25]. Clustering is also used as a preprocessing step for the supervised classification [26].

In [27], Naranjo et al. use various clustering approaches including the k-means model for automatic classification of traffic incidents. The approach proposed in [28] uses the k-means clustering model for the optimal scheduling of public transportation. Sesham et al. [29] use factor analysis methods in a combination with the k-means clustering for detecting cluster structures in transportation data obtained from the interview survey. Such data include the geographic information (home addresses) and general route information. The use of GPS sensors [30] for collecting traffic data provides us with large data arrays for such problems as the travel time prediction, traffic condition recognition [31], etc.

The k-means problem can be classified as a continuous location problem [32,33]: it is aimed at finding the optimal location of centroids in a continuous space.

If we replace squared distances with distances in (1), we deal with the very similar continuous k-median location problem [34] which is also a popular clustering model [35]. The k-medoids [36,37] problem is its discrete version where cluster centers must be selected among the data vectors only, which allows us to calculate the distance matrix in advance [38]. The k-median problem was also formulated as a discrete location problem [39] on a graph. The similarity of these NP-hard problems [40,41] enables us to use similar approaches to solve them. In the early attempts to solve the k-median problem (its discrete version) by exact methods, researchers used a branch and bound algorithm [42–44] for solving very small problems.

Metaheuristic approaches, such as genetic algorithms [45], are aimed at finding the global optimum. However, in large-scale instances, such approaches require very significant computational costs, especially if they are adapted to solving continuous problems [46].

With regard to discrete optimization problems, local search methods, which include Lloyd's procedure, have been developed since the 1950s [47–50]. These methods have been successfully used to solve location problems [51,52]. The progress of local search methods is associated with both new algorithmic schemes and new theoretical results in the field of local search [50].

A standard local search algorithm starts with some initial solution S and goes to a neighboring solution if this solution turns out to be superior. Moreover, finding the set of neighbor solutions $n(S)$ is the key issue. Elements of this set are formed by applying a certain procedure to a solution S . At each local search step, the neighborhood function $n(S)$ specifies the set of possible search directions. Neighborhood functions can be very diverse, and the neighborhood relation is not always symmetric [53,54].

For a review of heuristic solution techniques applied to k-means and k-median problems, the reader can refer to [32,55,56]. Brimberg, Drezner, and Mladenovic and Salhi [57–59] presented local search approaches including the variable neighborhood search (VNS) and concentric search. In [58], Drezner et al. proposed heuristic procedures including the genetic algorithm (GA), for rather small

data sets. Algorithms for finding the initial solution for the Lloyd's procedure [60,61] are aimed at improving the average resulting solution. For example, in [62], Bhusare et al. propose an approach to spread the initial centroids uniformly so that the distance among them is as far as possible. The most popular kmeans++ initialization method introduced by Arthur and Vassilvitskii [60] is a probabilistic implementation of the same idea. An approach proposed by Yang and Wang [63] improves the traditional k-means clustering algorithm by choosing initial centroids with a min-max similarity. Gu et al. [7] provide a density-based initial cluster center selection method to solve the problem of outliers. Such smart initialization algorithms reduce the search area for local search algorithms in multi-start modes. Nevertheless, they do not guarantee an optimal or near optimal solution of the problem (1).

Many authors propose approaches based on reducing the amount of data [64]: simplification of the problem by random (or deterministic) selection of a subset of the initial data set for a preliminary solution of the k-means problem, and using these results as an initial solution to the k-means algorithm on the complete data set [65–67]. Such aggregation approaches, summarized in [68], as well as reducing the number of the data vectors [69], enable us to solve large-scale problems within a reasonable time. However, such approaches lead to a further reduction in accuracy. Moreover, many authors [70,71] name their algorithm “exact” which does not mean the ability to achieve an exact solution of (1). In such algorithms, the word “exact” means the exact adherence to the scheme of the Lloyd's procedure, without any aggregation, sampling, and relaxation approaches. Thus, such algorithms may be faster than the Lloyd's procedure due to the use of triangle inequality, storing the results of distance calculations in multidimensional data sets or other tricks [72], however they are not intended to get the best value of (1). In our research, aimed at obtaining the most precise solutions, we consider only the methods which estimate the objective function (1) directly, without aggregation or approximation approaches.

The main idea of the variable neighborhood search algorithms proposed by Hansen and Mladenovic [73–75] is the alternation of neighborhood functions $n(S)$. Such algorithms include Lloyd's procedure, which alternates finding a locally optimal solution of a continuous optimization problem (3) with a solution of a combinatorial problem (4). However, as applied to the k-means problem, the VNS class traditionally involves more complex algorithms.

The VNS algorithms are used for a wide variety of problems [3,76,77] including clustering [78] and work well for solving k-means and similar problems [50,79–82].

Agglomerative and dissociative procedures are separate classes of clustering algorithms. Dissociative (divisive) procedures [83] are based on splitting clusters into smaller clusters. Such algorithms are commonly used for small problems due to their high computing complexity [83–85], most often in hierarchical clustering models. The agglomerative approach is the most popular in hierarchical clustering, however, it is also applied in other models of cluster analysis. Agglomerative procedures [86–90] combine clusters sequentially, i.e., in relation to the k-means problem, they sequentially remove centroids. The elements of the clusters, related to the removed centroids, are redistributed among the remaining clusters. The greedy strategies are used to decide which clusters are most similar to be merged together [3] at each iteration of the agglomerative procedure. An agglomerative procedure starts with some solution S containing an excessive number of centroids and clusters $k + r$, where integer r is known in advance or chosen randomly. The r value (number of excessive centroids in the temporary solution) is the most important parameter of the agglomerative procedure. Some algorithms, including those based on the k-means model [91], involve both the agglomerative and dissociative approaches. Moreover, such algorithms are not aimed at achieving the best value of the objective function (1), and their accuracy is not high in this sense.

1.3. Research Gap

Many transportation and other problems (e.g., clustering problems related to computer vision) require online computation within a fixed time. As mentioned above, Lloyd's procedure, the most popular k-means clustering algorithm, is rather fast. Nevertheless, for specific data sets including

geographic/geometrical data, this algorithm results in a solution which is very far from the global minimum of the objective function (1), and the multi-start operation mode does not improve the result significantly. More accurate k-means clustering methods are much slower. Nevertheless, recent advances in high-performance computing and the use of massively parallel systems enable us to work through a large amount of computation using the Lloyd's procedure embedded into more complex algorithmic schemes. Thus, the demand for clustering algorithms that compromise on the time spent for computations and the resulting objective function (1) value is apparent. Nevertheless, in some cases, when solving problem (1), it is required to obtain a result (a value of the objective function) within a limited fixed time, which would be difficult to improve on by known methods without a significant increase in computational costs. Such results are required if the cost of error is high, as well as for evaluating faster algorithms, as reference solutions.

Agglomerative procedures, despite their relatively high computational complexity, can be successfully integrated into more complex search schemes. They can be used as a part of the crossover operator of genetic algorithms [46,88] and as a part of the VNS algorithms. Moreover, such algorithms are a compromise between the solution accuracy and time costs. In this article, by accuracy, we mean exclusively the ability of the algorithm (solver) to obtain the minimum values of the objective function (1).

The use of VNS algorithms, that search in the neighborhoods, formed by applying greedy agglomerative procedures to a known (current) solution S , enables us to obtain good results in a fixed time acceptable for interactive modes of operation. The selection of such procedures, their sequence and their parameters remained an open question. The efficiency of such procedures has been experimentally shown on some test and practical problems. Various versions of VNS algorithms based on greedy agglomerative procedures differ significantly in their results which makes such algorithm difficult to use in practical problems. It is practically impossible to forecast the relative performance of a specific VNS algorithm based on such generalized numerical features of the problem as the sample size and the number of clusters. Moreover, the efficiency of such procedures depends on their parameters. However, the type and nature of this dependence has not been studied.

1.4. Our Contribution

In this article, we systematize approaches to the construction of search algorithms in neighborhoods, formed by the use of greedy agglomerative procedures.

In this work, we proceeded from the following assumptions:

- (a) The choice of parameter r value (the number of excessive centroids, see above) of the greedy agglomerative heuristic procedure significantly affects the efficiency of the procedure.
- (b) Since it is hardly possible to determine the optimal value of this parameter based on such numerical parameters of the k-means problem as the number of data vectors and the number of clusters, reconnaissance (exploratory) search with various values of r can be useful.
- (c) Unlike the well-known VNS algorithms that use greedy agglomerative heuristic procedures with an increasing value of the parameter r , a gradual decrease in the value of this parameter may be more effective.

Based on these assumptions, we propose a new VNS algorithm involving greedy agglomerative procedures for the k-means problem, which, by adjusting the initial r parameter of such procedures, enables us to obtain better results in a fixed time which exceed the results of known VNS algorithms. Due to self-adjusting capabilities, such an algorithm should be more versatile, which should increase its applicability to a wider range of problems in comparison with known VNS algorithms based on greedy agglomerative procedures.

1.5. Structure of this Article

The rest of this article is organized as follows. In Section 2, we present an overview of the most common local search algorithms for k-means and similar problems, and introduce the notion

of neighborhoods $SWAP_r$ and $GREEDY_r$. It is shown experimentally that the search result in these neighborhoods strongly depends on the neighborhood parameter r (the number of simultaneously alternated or added centroids). In addition, we present a new VNS algorithm which performs the local search in alternating $GREEDY_r$ neighborhoods with the decreasing value of r and its initial value estimated by a special auxiliary procedure. In Section 3, we describe our computational experiments with the new and known algorithms. In Section 4, we consider the applicability of the results on the adjustment of the $GREEDY_r$ neighborhood parameter in algorithmic schemes other than VNS, in particular, in evolutionary algorithms with a greedy agglomerative crossover operator. The conclusions are given in Section 5.

2. Materials and Methods

For constructing a more efficient algorithm (solver), we used a combination of such algorithms as Lloyd's procedure, greedy agglomerative clustering procedures, and the variable neighborhood search. The most computationally expensive part of this new algorithmic construction, Lloyd's procedure, was implemented on graphic processing units (GPU).

2.1. The Simplest Approach

Lloyd's procedure, the simplest and most popular algorithm for solving the k-means problem, is described as follows (see Algorithm 1).

Algorithm 1. *Lloyd(S)*

Require: Set of initial centroids $S = \{X_1, \dots, X_k\}$. If S is not given, then the initial centroids are selected randomly from the set of data vectors $\{A_1, \dots, A_N\}$.

repeat

1. For each centroid $X_j, j = \overline{1, k}$, define its cluster in accordance with (2); // I.e. assign each data vector to the nearest centroid
2. For each cluster $C_j, j = \overline{1, k}$, calculate its centroid as follows:

$$X_j = \frac{\sum_{i \in \{1, N\}: A_i \in C_j} A_i}{|C_j|}.$$

until all centroids stay unchanged.

Formally, the k-means problem in its formulation (1) or (3) is a continuous optimization problem. With a fixed composition of clusters C_j , the optimal solution is found in an elementary way, see Step 2 in Algorithm 1, and this solution is the local optimum of the problem in terms of the continuous optimization theory, i.e., local optimum in the ε -neighborhood. A large number of such optima forces the algorithm designers to systematize their search in some way. The first step of Lloyd's algorithm solves a simple combinatorial optimization problem (3) on the redistribution of data vectors among clusters, that is, it searches in the other neighborhood.

The simplicity of Lloyd's procedure enables us to apply it to a wide range of problems, including face detection, image segmentation, signal processing and many others [92]. Frackiewicz et al. [93] presented a color quantization method based on downsampling of the original image and k-means clustering on a downsampled image. The k-means clustering algorithm used in [94] was proposed for identifying electrical equipment of a smart building. In many cases, researchers do not distinguish between the k-means model and the k-means algorithm, as Lloyd's procedure is also called. Nevertheless, the result of Lloyd's procedure may differ from the results of other more advanced algorithms many times in the objective function value (1). For finding a more accurate solution, a wide range of heuristic methods were proposed [55]: evolutionary and other bio-inspired algorithms, as well as local search in various neighborhoods.

Modern scientific literature offers many algorithms to speed up the solution of the k-means problem. Algorithm named k-indicators [95] promoted by Chen et al. is a semi-convex-relaxation algorithm for approximate solution of big-data clustering problems. In the distributed implementation of the k-means algorithm proposed in [96], the algorithm considers a set of agents, each of which is equipped with a possibly high-dimensional piece of information or set of measurements. In [97,98], the researchers improved algorithms for the data streams. In [99], Hedar et al. present a hierarchical k-means method for better clustering performance in the case of big data problems. This approach enables us to mitigate the poor scaling behavior with regard to computing time and memory requirements. Fast adaptive k-means subspace clustering algorithm with an adaptive loss function for high-dimensional data was proposed by Wang et al. [100]. Nevertheless, the usage of the massively parallel systems is the most efficient way to achieve the most significant acceleration of computations, and the original Lloyd's procedure (Algorithm 1) can be seamlessly parallelized on such systems [101,102].

Metaheuristic approaches for the k-means and similar problems include genetic algorithms [46,103,104], the ant colony clustering hybrid algorithm proposed in [105], particle swarm optimization algorithms [106]. Almost all of these algorithms in one way or another use the Lloyd's procedure or other local search procedures. Our new algorithm (solver) is not an exception.

2.2. Local Search in SWAP Neighborhoods

Local search algorithms differ in forms of neighborhood function $n(S)$. A local minimum in one neighborhood may not be a local minimum in another neighborhood [50]. The choice of a neighborhood of lower cardinality leads to a decrease in the complexity of the search step, however, a wider neighborhood can lead to a better local minimum. We have to find a balance between these conflicting requirements [50].

A popular idea when solving k-means, k-medoids, k-median problems is to search for a better solution in SWAP neighborhoods. This idea was realized, for instance, in the J-means procedure [80] proposed by Hansen and Mladenovic, and similar I-means algorithm [107]. In SWAP neighborhoods, the set $n(S)$ is the set of solutions obtained from S by replacing one or more centroids with some data vectors.

Let us denote the neighborhood, where r centroids must be simultaneously replaced, by $\text{SWAP}_r(S)$. The SWAP_r neighborhood search can be regular (all possible substitutions are sequentially enumerated), as in the J-means algorithm, or randomized (centroids and data vectors for replacement are selected randomly). In both cases, the search in the SWAP neighborhood always alternates with the Lloyd's procedure: if an improved solution is found in the SWAP neighborhood, the Lloyd's procedure is applied to this new solution, and then the algorithm returns to the SWAP neighborhood search. Except for very small problems, regular search in SWAP neighborhoods, with the exception of the SWAP_1 neighborhood and sometimes SWAP_2 , is almost never used due to the computational complexity: in each of iterations, all possible replacement options must be tested. A randomized search in SWAP_r neighborhoods can be highly efficient for sufficiently large problems, which can be demonstrated by the experiment described below. Herewith, the correct choice of r is of great importance.

As can be seen on Figure 1, for various problems from the clustering benchmark repository [108,109], the best results are achieved with different values of r , although in general, such a search provides better results in comparison with Lloyd's procedure. Our computational experiments are described in detail in Sections 2.5–2.7.

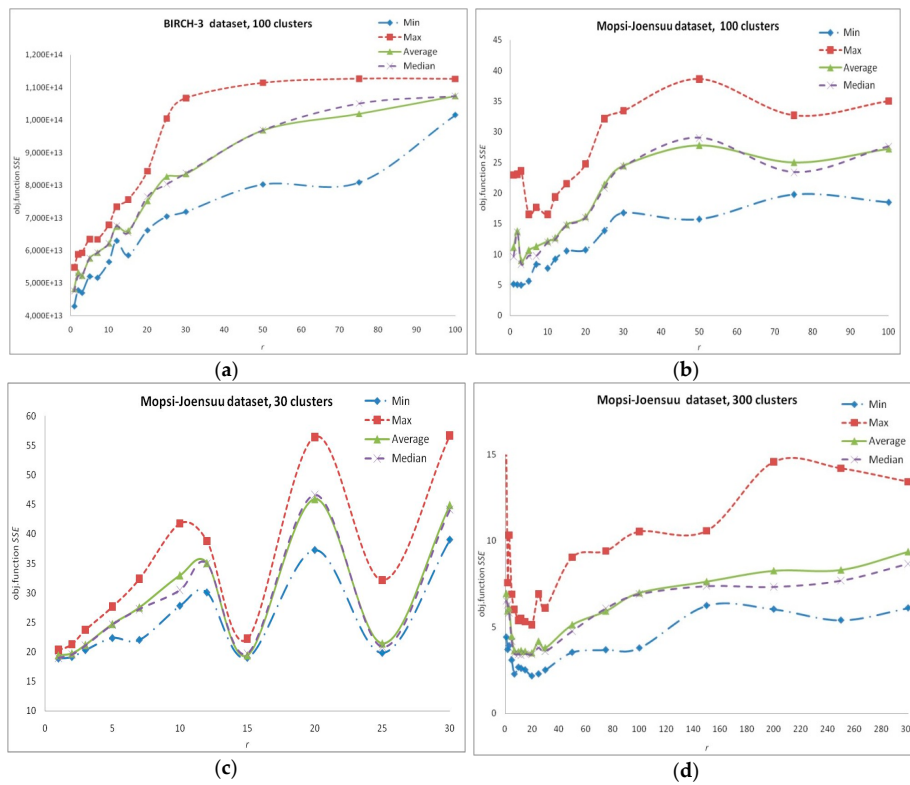


Figure 1. Search in SWAP r neighborhoods. Dependence of the result on r : (a) BIRCH3 data set, 100 clusters, 10^5 data vectors, time limitation 10 s; (b–d) Mopsi-Joensuu data set, 30, 100 and 300 clusters, 6014 data vectors, time limitation 5 s.

2.3. Agglomerative Approach and GREEDY $_r$ Neighborhoods

When solving the k-means and similar problems, the agglomerative approach is often successful. In [86], Sun et al. propose a parallel clustering method based on MapReduce model which implements the information bottleneck clustering (IBC) idea. In the IBC and other agglomerative clustering algorithms, clusters are sequentially removed one-by-one, and objects are redistributed among the remaining clusters. Alp et al. [88] presented a genetic algorithm for facility location problems, where evolution is facilitated by a greedy agglomerative heuristic procedure. A genetic algorithm with a faster greedy heuristic procedure for clustering and location problems was also proposed in [90]. In [46], two genetic algorithm approaches with different crossover procedures are used to solve k-median problem in continuous space.

Greedy agglomerative procedures can be used as independent algorithms, as well as being embedded into genetic operators [110] or VNS algorithms [79]. The basic greedy agglomerative procedure for the k-means problem can be described as follows (see Algorithm 2).

Algorithm 2. BasicGreedy(S)

Require: Set of initial centroids $S = \{X_1, \dots, X_K\}$, $K > k$, required final number of centroids k .

$S \leftarrow \text{Lloyd}(S)$;

while $|S| > k$ **do**

for $i = 1, \overline{K}$ **do**

$F_i \leftarrow \text{SSE}(S \setminus \{X_i\})$;

end for

 Select a subset $S' \subset S$ of r_{remove} centroids with the minimum values of the corresponding variables F_i ; // By default, $r_{\text{remove}} = 1$.

$S \leftarrow \text{Lloyd}(S \setminus S')$;

end while.

In its most commonly used version, with $r_{\text{remove}} = 1$, this procedure is rather slow for large-scale problems. It tries to remove the centroids one-by-one. At each iteration, it eliminates such centroids that their elimination results in the least significant increase in the SSE value. Further, this procedure involves the Lloyd's procedure which can be also slow in the case of rather large problems with many clusters. To improve the performance of such a procedure, the number of simultaneously eliminated centroids can be calculated as $r_{\text{remove}} = \max\{1, (|S| - k) \cdot r_{\text{coef}}\}$. In [90], Kazakovtsev and Antamoshkin used the elimination coefficient value $r_{\text{coef}} = 0.2$. This means that at each iteration, up to 20% of the excessive centroids are eliminated, and such values are proved to make the algorithm faster. In this research, we use the same value.

In [79,90,110], the authors embed the *BasicGreedy()* procedure into three algorithms which differ in r value only. All of these algorithms can be described as follows (see Algorithm 3):

Algorithm 3. *Greedy* (S, S_2, r)

Require: Two sets of centroids $S, S_2, |S| = |S_2| = k$, the number of centroids r of the solution S_2 which are used to obtain the resulting solution, $r \in \{1, k\}$.

For $i = 1, n_{\text{repeats}}$ **do**

1. Select a subset $S' \subset S_2 : |S'| = r$.
2. $S' \leftarrow \text{BasicGreedy}(S \cup S')$;
3. **if** $\text{SSE}(S') < \text{SSE}(S)$ **then** $S \leftarrow S'$ **end if**;

end for

return S .

Such procedures use various values of r from 1 up to k . If $r = 1$ then the algorithm selects a subset (actually, a single element) of S_2 regularly: $\{X_1\}$ in the first iteration, $\{X_2\}$ in the second one, etc. In this case, $n_{\text{repeats}} = k$. If $r = k$ then obviously $S' = S_2$, and $n_{\text{repeat}} = 1$. Otherwise, r is selected randomly, $r \in \{2, k-1\}$, and n_{repeats} depends on r : $n_{\text{repeats}} = \max\{1, \lfloor k/r \rfloor\}$.

If the solution S_2 is fixed, then all possible results of applying the *Greedy*(S, S_2, r) procedure form a neighborhood of the solution S , and S_2 as well as r are parameters of such a neighborhood. If S_2 is a randomly chosen locally optimal solution obtained by *Lloyd*(S_2') procedure applied to a randomly chosen subset $S_2' \subset \{A_1, \dots, A_N\}$, $|S_2'| = k$, then we deal with a randomized neighborhood.

Let us denote such a neighborhood by $\text{GREEDY}_r(S)$. Our experiments in Section 3 demonstrate that the obtained result of the local search in GREEDY_r neighborhoods strongly depends on r .

2.4. Variable Neighborhood Search

The dependence of the local search result on the neighborhood selection reduces if we use a certain set of neighborhoods and alternate them. This approach is the basis for VNS algorithms. The idea of alternating neighborhoods is easy to adapt to various problems [76–78] and highly efficient, which makes it very useful for solving NP-hard problems including clustering, location, and vehicle routing problems. In [111,112], Brimberg and Mladenovic and Miskovic et al. used the VNS for solving various facility location problems. Cranic et al. [113] as well as Hansen and Mladenovic [114] proposed and developed a parallel VNS algorithm for the k-median problem. In [115], a VNS algorithm was used for a vehicle routing and driver scheduling problems by Wen et al.

The ways of neighborhood alternation may differ significantly. Many VNS algorithms are not even classified by their authors as VNS algorithms. For example, the algorithm in [57] alternates between discrete and continuous problems: when solving a discrete problem, the set of local optima is replenished, and then such local optima are chosen as elements of the initial solution of the continuous problem. A similar idea of the recombinator k-means algorithm was proposed by C. Baldassi [116]. This algorithm restarts the k-means procedure, using the results of previous runs as a reservoir of candidates for the new initial solutions, exploiting the popular k-means++ seeding

algorithm to piece them together into new, promising initial configurations. Thus, the k-means search alternates with the discrete problem of finding an optimal initial centroid combination.

VNS class includes a very efficient abovementioned J-Means algorithm [80], which alternates search in a SWAP neighborhood and the use of Lloyd's procedure. Even when searching only in the $SWAP_1$ neighborhood, the J-Means results can be many times better than the results of Lloyd's procedure launched in the multi-start mode, as shown in [62,97].

In [50], Kochetov et al. describe such basic schemes of VNS algorithms as variable neighborhood descent (VND, see Algorithm 4) [117] and randomized Variable Neighborhood Search (RVNS, see Algorithm 5) [50].

Algorithm 4. VND(S)

Require: Initial solution S , selected neighborhoods $n_l, l = \overline{1, l_{max}}$.

repeat

$l \leftarrow 1$;

while $l \leq l_{max}$ **do**

 search for $S' \in n_l(S) : f(S') = \min\{f(Y) | Y \in n_l(S)\}$;

if $f(S') < f(S)$ **then** $S \leftarrow S'$; $l \leftarrow 1$ **else** $l \leftarrow l + 1$ **end if**;

end while;

until the stop conditions are satisfied.

Algorithm 5. RVNS(S)

Require: Initial solution S , selected neighborhoods $n_l, l = \overline{1, l_{max}}$.

repeat

$l \leftarrow 1$;

While $l \leq l_{max}$ **do**

 select randomly $S' \in n_l(S)$;

if $f(S') < f(S)$ **then** $S \leftarrow S'$; $l \leftarrow 1$ **else** $l \leftarrow l + 1$ **end if**;

end while;

until the stop conditions are satisfied.

Algorithms of the RVNS scheme are more efficient when solving large-scale problems [50], when the use of deterministic VND requires too large computational costs per each iteration. In many efficient algorithms, $l_{max} = 2$. For example, the J-Means algorithm combines a SWAP neighborhood search with Lloyd's procedure.

As a rule, algorithm developers propose to move from neighborhoods of lower cardinality to wider neighborhoods. For instance, in [79], the authors propose a sequential search in the neighborhoods $GREEDY_1 \rightarrow GREEDY_{random} \rightarrow GREEDY_k \rightarrow GREEDY_1 \rightarrow \dots$. Here, $GREEDY_{random}$ is a neighborhood with randomly selected $r \in \overline{2, k-1}$. In this case, the initial neighborhood type has a strong influence on the result [79]. However, the best initial value of parameter r is hardly predictable.

In this article, we propose a new RVNS algorithm which involves $GREEDY_r$ neighborhood search with a gradually decreasing r and automatic adjustment of the initial r value. Computational experiments show the advantages of this algorithm in comparison with the algorithms searching in SWAP neighborhoods as well as in comparison with known search algorithms with $GREEDY_r$ neighborhoods.

2.5. New Algorithm

A search in a $GREEDY_r$ neighborhood with a fixed r values, on various practical problems listed in the repositories [108,109,118], shows that the result (the value of the objective function) essentially depends on r , and this dependence differs for various problems, even if the problems have similar basic numerical characteristics, such as the number of data vectors N , their dimension d , and the number of clusters k . The results are shown on Figures 2 and 3. At the same time, our experiments show that at the first iterations, the use of Algorithm 3 almost always leads to an improvement in the SSE value,

and then the probability of such a success decreases. Moreover, the search in neighborhoods with large r values stops giving improving results sooner, while the search in neighborhoods with small r , in particular, with $r = 1$, enables us to obtain the improved solutions during a longer time. The search in the GREEDY₁ neighborhood corresponds to the adjustment of individual centroid positions. Thus, the possible decrement of the objective function value is not the same for different values of r .

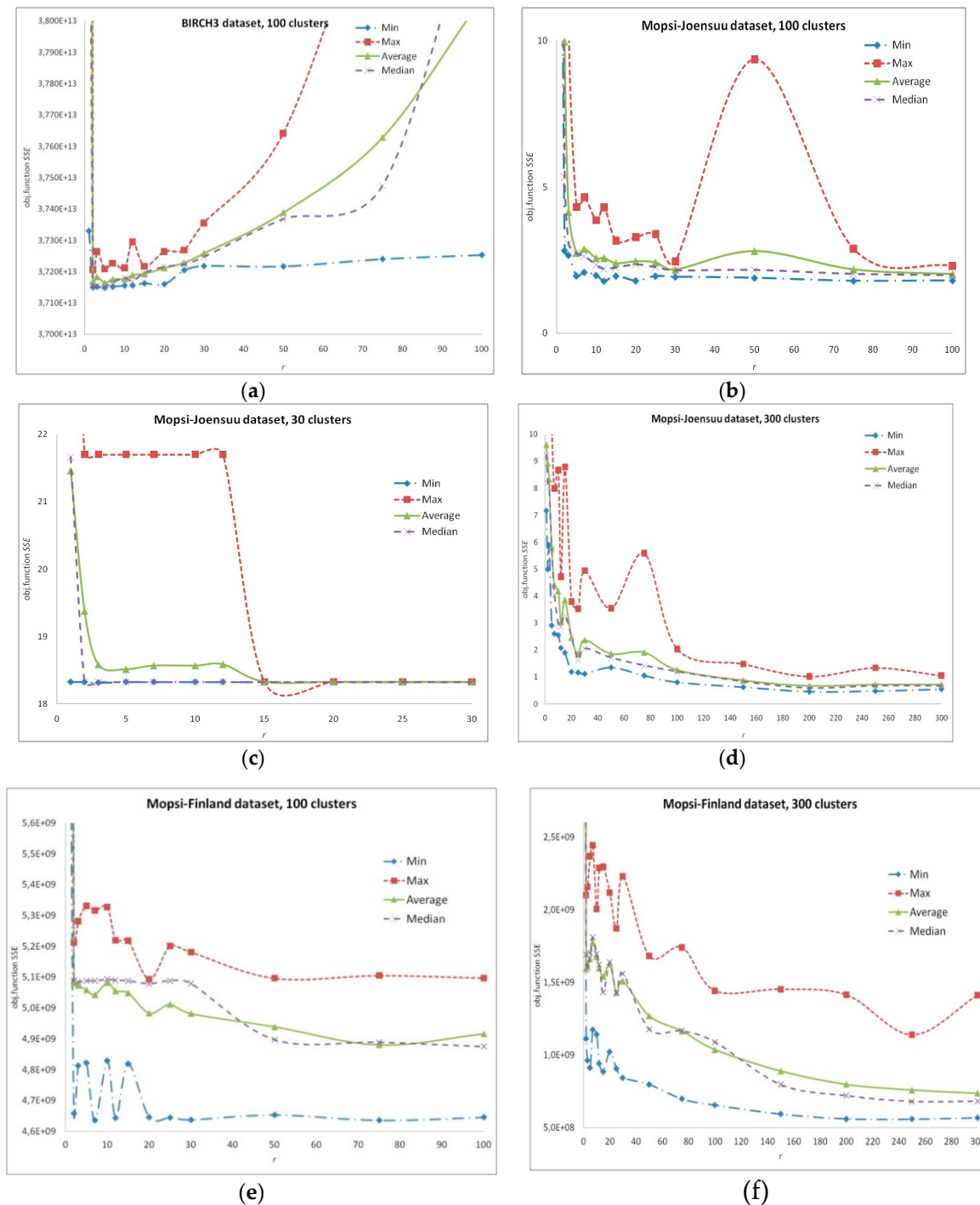


Figure 2. Search in GREEDY_r neighborhoods. Dependence of the result on r : (a) BIRCH3 data set, 100 clusters, 10^5 data vectors, time limitation 10 s; (b–d) Mopsi-Joensuu data set, 30, 100 and 300 clusters, 6014 data vectors, time limitation 5 s; (e–f) Mopsi-Finland data set, 100 and 300 clusters, 13,467 data vectors, time limitation 5 s.

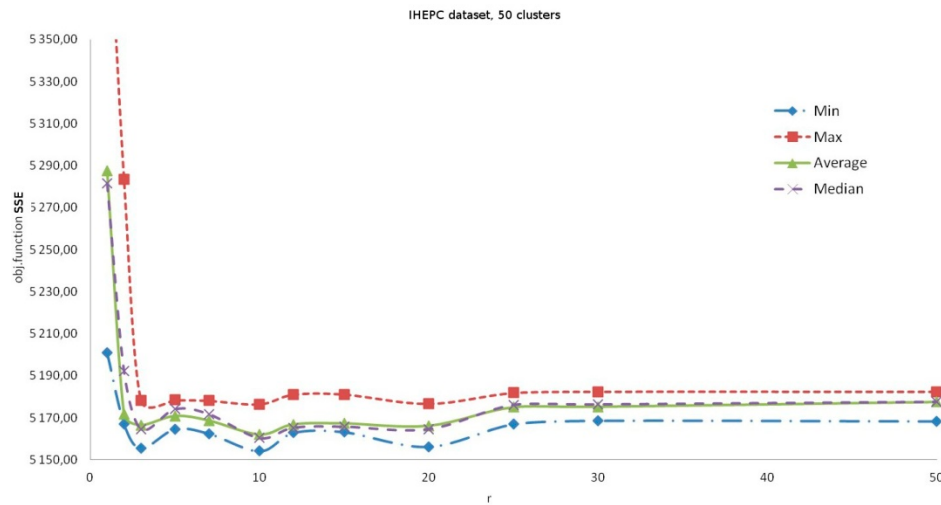


Figure 3. Search in GREEDY_r neighborhoods. Dependence of the result on r : Individual Household Electric Power Consumption (IHEPC) data set, 50 clusters, 2,075,259 data vectors, time limitation 5 min.

We propose the following sequence of neighborhoods: GREEDY _{r_0} → GREEDY _{r_1} → GREEDY _{r_2} → ... → GREEDY₁ → GREEDY _{k} → ... Here, r values gradually decrease: $r_0 > r_1 > r_2 \dots$. After reaching $r = 1$, the search continues in the GREEDY _{k} neighborhood, and after that the value of r starts decreasing again. Moreover, the r value fluctuates within certain limits at each stage of the search.

This algorithm can be described as follows (Algorithm 6).

Algorithm 6. DecreaseGreedySearch(S, r_0)

Require: Initial solution S , initial $r = r_0 \in \{1, k\}$.
 select randomly $S_2 \subset \{A_1, \dots, A_N\}, |S_2| = k; S_2 \leftarrow \text{Lloyd}(S_2);$
repeat
 $n_{\text{repeats}} \leftarrow \max\{1, \lfloor k/r \rfloor\};$
 for $i = 1, n_{\text{repeats}}$ **do**
 1. select randomly $r' \in \left\{ \overline{\max\{1, \lfloor \frac{r_0}{2} \rfloor\}}, r_0 \right\};$
 2. $S' \leftarrow \text{Greedy}(S, S_2, r');$
 3. **if** $\text{SSE}(S') < \text{SSE}(S)$ **then** $S \leftarrow S'$ **end if**;
endfor;
 select randomly $S_2 \subset \{A_1, \dots, A_N\}, |S_2| = k; S_2 \leftarrow \text{Lloyd}(S_2);$
if Steps 1–3 have not changed S
 then
if $r = 1$ **then** $r_0 \leftarrow k$ **else** $r_0 \leftarrow \max\{1, \lfloor \frac{r}{2} \rfloor - 1\}$ **end if**;
end if;
until the stop conditions are satisfied (time limitation).

Genetic algorithms with greedy agglomerative heuristics are known to perform better than VNS algorithms with sufficient computation time [79,90] which results in better SSE values. Despite this, the limited time and computational complexity of the Greedy() procedure as a genetic crossover operator leads to a situation when genetic algorithms may have enough time to complete a very limited number of crossover operations and often only reach the second or third generation of solutions. Under these conditions, VNS algorithms are a reasonable compromise of the computation cost and accuracy.

The choice of the initial value of parameter r_0 is highly important. Such a choice is quite simply carried out by a reconnaissance search with different r_0 values. The algorithm with such an automatic adjustment of the parameter r_0 by performing a reconnaissance search is described as follows (Algorithm 7).

Algorithm 7. *AdaptiveGreedy* (S) solver

Require: the number of reconnaissance search iterations n_{recon} .
 select randomly $S \subset \{A_1, \dots, A_N\}, |S| = k; S \leftarrow Lloyd(S);$
for $i = 1, n_{recon}$ **do**
 select randomly $S_i \subset \{A_1, \dots, A_N\}, |S_i| = k; S_i \leftarrow Lloyd(S_i);$
end for;
 $r \leftarrow k;$
repeat
 $S_{r''} \leftarrow S; n_{repeats} \leftarrow \max\{1, \lceil k/r \rceil\};$
 for $i = 1, n_{repeats}$ **do**
 for $i = 1, n_{repeats}$ **do**
 $S' \leftarrow Greedy(S_{r''}, S_i, r);$ **if** $SSE(S') < SSE(S_{r''})$ **then** $S_{r''} \leftarrow S'$ **end if**;
 end for;
 end for;
 $r \leftarrow \max\{1, \lceil \frac{r}{2} \rceil - 1\};$
until $r = 1$;
 select the value r with minimum value of $SSE(S_{r''})$;
 $r_0 \leftarrow \min\{1.5r, k\};$
 DecreaseGreedySearch($S_{r''}, r_0$).

Results of computational experiments described in the next Section show that our new algorithm, which sequentially decreases the value of the parameter r_0 , has an advantage over the known VNS algorithms.

2.6. CUDA Implementation

The greedy agglomerative procedure (*BasicGreedy*) is computationally expensive. In Algorithm 2, the objective function calculation $F_T \leftarrow SSE(S \setminus \{X_i\})$ is performed more than $(K - k) \cdot k$ times in each iteration, and after that, *Lloyd*() procedure is executed. Therefore, such algorithms are traditionally considered as methods for solving comparatively small problems (hundreds of thousands of data points and hundreds of clusters). However, the rapid development of the massive parallel processing systems (GPUs) enables us to solve the large-scale problems with reasonable time expenses (seconds). Parallel (CUDA) implementation of the algorithms for the *Lloyd*() procedure is known [101,102], and we used this approach in our experiments.

Graphic processing units (GPUs) accelerate computations with the use of multi-core computing architecture. The CUDA (compute unified device architecture) is the most popular programming platform which enables us to use general-purpose programming languages (e.g., C++) for compiling GPU programs. The programming model uses the single instruction multiple thread (SIMT) principle [119]. We can declare a function in the CUDA program a “kernel” function and run this function on the steaming multiprocessors. The threads are divided into blocks. Several instances of a kernel function are executed in parallel on different nodes (blocks) of a computation grid. Each thread can be identified by special *threadIdx* variable. Each thread block is identified by *blockIdx* variable. The number of threads in a block is identified by *blockDim* variable. All these variables are 3-dimensional vectors (dimensions x, y, z). Depending on the problem solved, the interpretation of these dimensions may differ. For processing 2D graphical data, x and y are used for identifying pixel coordinates.

The most computationally expensive part of Lloyd’s procedure is distance computation and comparison (Step 1 of Algorithm 1). This step can be seamlessly parallelized if we calculate distances from each individual data vector in a separate thread. Thus, *threadIdx.x* and *blockIdx.x* must indicate a data vector. The same kernel function prepares data needed for centroid calculation (Step 2 of Algorithm 1). Such data are the sum of data vector coordinates in a specific cluster $sum_j = \sum_{i \in \{1, N\}: A_i \in C_j} A_i$ and the cardinality of the cluster $counter_j = |C_j|$. Here, j is the cluster number. Variable sum_j is a vector (1-dimensional array in program realization).

To perform Step 1 of Algorithm 1 on a GPU, after initialization $sum_j \leftarrow 0$ and $counter_j \leftarrow 0$, the following procedure (Algorithm 8) runs on $(N + blockDim.x) / blockDim.x$ nodes of computation grid, with $blockDim.x$ threads in each block (in our experiments, $blockDim.x = 512$):

Algorithm 8. CUDA kernel implementation of Step 1 in Lloyd's procedure (Algorithm 1)

```

i ← blockDim.x · blockDim.x + threadIdx.x;
if i > N then return end if;
 $D_{nearest} \leftarrow +\infty$ ; // distance from  $A_i$  to the nearest centroid
for  $j = \overline{1, k}$  do
    if  $\|A_j - X_i\| < D_{nearest}$  then
         $D_{nearest} \leftarrow \|A_j - X_i\|$ ;
         $n \leftarrow j$ ;
    end if
end for;
 $sum_n \leftarrow sum_n + A_n$ ;
 $counter_n \leftarrow counter_n + 1$ ;
 $SSE \leftarrow SSE + D_{nearest}^2$  // objective function adder

```

If sum_j and $counter_j$ are pre-calculated for each cluster then Step 2 of Algorithm 1 is reduced to a single arithmetic operation for each cluster: $X_j = sum_j / counter_j$. If the number of clusters is not huge, this operation does not take significant computation resources. Nevertheless, its parallel implementation is even simpler: we organize k threads, and each thread calculates X_j for an individual cluster. Outside Lloyd's procedure, we use Algorithm 8 for SSE value estimation (variable SSE must be initialized by 0 in advance).

The second computationally expensive part of the *BasicGreedy()* algorithm is estimation of the objective function value after eliminating a centroid [120]: $F_{i'} = SSE(S \setminus \{X_i\})$. Having calculated $SSE(S)$, we may calculate as $SSE(S \setminus \{X_i\})$ as

$$F_{i'} = SSE(S \setminus \{X_i\}) = SSE(S) + \sum_{l=1}^N \Delta D_l \quad (5)$$

where

$$\Delta D_l = \begin{cases} 0, & A_l \notin C_i, \\ \left(\min_{j \in \overline{1, k}, j \neq i} \|X_j - A_l\| \right)^2 - \|X_i - A_l\|^2, & A_l \in C_i. \end{cases}$$

For calculating (5) on a GPU, after initializing $F_i \leftarrow SSE(S)$, the following kernel function (Algorithm 9) runs for each data vector.

Algorithm 9. CUDA kernel implementation of calculating $F_i \leftarrow SSE(S \setminus \{X_i\})$ in *BasicGreedy* procedure (Algorithm 2)

```

Require: index i of centroid being eliminated.
l ← blockDim.x · blockDim.x + threadIdx.x;
if l > N then return end if;
 $D_{nearest} \leftarrow +\infty$ ; // distance from  $A_l$  to the nearest centroid except  $X_i$ 
for  $j = \overline{1, k}$  do
    if  $l \neq i$  and  $A_j - X_i < D_{nearest}$  then
         $D_{nearest} \leftarrow \|A_j - X_i\|$ ;
    end if
end for;
 $F_i \leftarrow F_i + D_{nearest}^2 - \|X_i - A_l\|^2$ ;

```

All distance calculations for *GREEDY*, neighborhood search are performed by Algorithms 8 and 9. A similar kernel function was used for accelerating the local search in *SWAP* neighborhoods. In this function, after eliminating a centroid, a data point is included in solution *S* as a new centroid.

All other parts of new and known algorithms were implemented on the CPU.

2.7. Benchmarking Data

In all our experiments, we used the classic data sets from the UCI Machine Learning and Clustering basic benchmark repositories [108,109,118]:

- Individual household electric power consumption (IHEPC)—energy consumption data of households during several years (more than 2 million data vectors, 7 dimensions), 0–1 normalized data, “date” and “time” columns removed;
- BIRCH3 [121]: one hundred of groups of points of random size on a plane (10^5 data vectors, 2 dimensions);
- S1 data set: Gaussian clusters with cluster overlap (5000 data vectors, 2 dimensions);
- Mopsi-Joensuu: geographic locations of users (6014 data vectors, 2 dimensions) in Joensuu city;
- Mopsi-Finland: geographic locations of users (13,467 data vectors, 2 dimensions) in Finland.

Mopsi-Joensuu and Mopsi-Finland are “geographic” data sets with a complex cluster structure, formed under the influence of natural factors such as the geometry of the city, transport communications, and urban infrastructure (Figure 4).

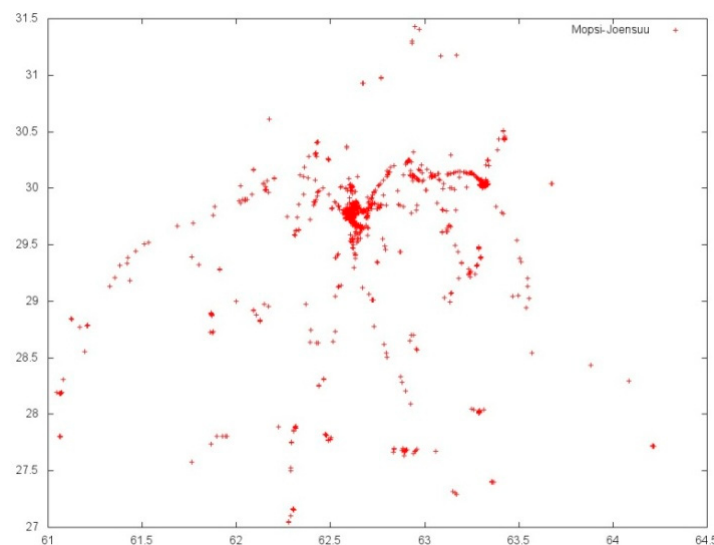


Figure 4. Mopsi-Joensuu data set visualization.

In our study, we do not take into account the true labeling provided by the data set (if it is known), i.e., the given predictions for known classes, and focus on the minimization of SSE only.

2.8. Computational Environment

For our computational experiments, we used the following test system: Intel Core 2 Duo E8400 CPU, 16GB RAM, NVIDIA GeForce GTX1050ti GPU with 4096 MB RAM, floating-point performance 2138 GFLOPS. This choice of the GPU hardware was made due to its prevalence, and also one of the best values of the price/performance ratio. The program code was written in C++. We used Visual C++ 2017 compiler embedded into Visual Studio v.15.9.5, NVIDIA CUDA 10.0 Wizards, and NVIDIA Nsight Visual Studio Edition CUDA Support v.6.0.0.

3. Results

For all data sets, 30 attempts were made to run each of the algorithms (see Tables 1 and A1, Tables A2–A11 in Appendix A).

Table 1. Comparative results for all data sets (best of known algorithms vs. new algorithm).

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs					<i>p</i> -Values and Statistical Significance of Difference in Results
	Min (Record)	Max (Worst)	Average	Median	Std.dev	
BIRCH3 data set. 10^5 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 10 s						
GREEDY ₂₀₀	1.30773×10^{13}	1.31172×10^{13}	1.30916×10^{13}	1.30912×10^{13}	1.08001×10^{10}	$p_t = 0.4098 \leftrightarrow$
AdaptiveGreedy	1.30807×10^{13}	1.31113×10^{13}	1.30922×10^{13}	1.30925×10^{13}	0.87731×10^{10}	$p_U = 0.2337 \leftrightarrow$
BIRCH3 data set. 10^5 data vectors in $\mathbb{R}^2$, $k = 100$ clusters, time limitation 10 s						
GREEDY ₅	3.71485×10^{13}	3.72087×10^{13}	3.71644×10^{13}	3.71518×10^{13}	2.22600×10^{10}	$p_t = 0.0701 \leftrightarrow$
AdaptiveGreedy	3.71484×10^{13}	3.72011×10^{13}	3.71726×10^{13}	3.71749×10^{13}	2.02784×10^{10}	$p_U = 0.1357 \leftrightarrow$
Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 5 s						
GH-VNS3	0.4321	0.6838	0.6024	0.6139	0.0836	$p_U = 0.00005 \uparrow$
GREEDY ₂₀₀	0.4555	1.0154	0.6746	0.5882	0.2163	$p_t < 0.00001 \uparrow$
AdaptiveGreedy	0.3128	0.6352	0.4672	0.4604	0.1026	
Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 100$ clusters, time limitation 5 s						
GREEDY ₁₀₀	1.8021	2.2942	2.0158	1.9849	0.1860	$p_t = 0.0910 \leftrightarrow$
GH-VNS3	1.7643	2.7357	2.0513	1.9822	0.2699	$p_U = 0.0042 \uparrow$
AdaptiveGreedy	1.7759	2.3265	1.9578	1.9229	0.1523	
Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 30$ clusters, time limitation 5 s						
GH-VNS1	18.3147	18.3255	18.3238	18.3253	0.0039	$p_t = 0.4118 \leftrightarrow$
AdaptiveGreedy	18.3146	18.3258	18.3240	18.3253	0.0037	$p_U = 0.2843 \leftrightarrow$
Mopsi- Finland data set.13,467 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 5 s						
GH-VNS3	5.33373×10^8	7.29800×10^8	5.74914×10^8	5.48427×10^8	5.05346×10^7	$p_t = 0.1392 \leftrightarrow$
AdaptiveGreedy	5.27254×10^8	7.09410×10^8	5.60867×10^8	5.38952×10^8	4.89257×10^7	$p_U = 0.0049 \uparrow$
Mopsi-Finland data set. 13,467 data vectors in $\mathbb{R}^2$, $k = 30$ clusters, time limitation 5 s						
GH-VNS3	3.42528×10^{10}	3.47955×10^{10}	3.43826×10^{10}	3.43474×10^{10}	1.02356×10^8	$p_t = 0.0520 \leftrightarrow$
AdaptiveGreedy	3.42528×10^{10}	3.47353×10^{10}	3.43385×10^{10}	3.43473×10^{10}	1.03984×10^8	$p_U = 0.0001 \uparrow$
S1 data set. 5000 data vectors in $\mathbb{R}^2$, $k = 15$ clusters, time limitation 1 second						
GH-VNS2	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.0000	$p_t = 0.5 \leftrightarrow$
AdaptiveGreedy	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.0000	$p_U = 0.5 \leftrightarrow$
S1 data set. 5000 data vectors in $\mathbb{R}^2$, $k = 50$ clusters, time limitation 1 second						
GH-VNS1	3.74310×10^{12}	3.76674×10^{12}	3.74911×10^{12}	3.74580×10^{12}	6.99859×10^9	$p_t = 0.3571 \leftrightarrow$
AdaptiveGreedy	3.74340×10^{12}	3.76313×10^{12}	3.74851×10^{12}	3.75037×10^{12}	5.56298×10^9	$p_U = 0.28434 \leftrightarrow$
IHEPC data set. 2,075,259 data vectors in $\mathbb{R}^7$, $k = 50$ clusters, time limitation 5 min						
GREEDY ₁₀	5154.2017	5176.4502	5162.0460	5160.4014	7.2029	$p_t = 0.008 \uparrow$
AdaptiveGreedy	5153.5640	5163.9316	5157.0822	5155.5198	3.6034	$p_U = 0.001 \uparrow$

Note: “ $\uparrow$ ”, “ $\downarrow$ ”: the advantage of the new algorithms over known algorithms is statistically significant (“ $\uparrow$ ” for *t*-test and “ $\downarrow$ ” for Mann–Whitney U test), “ $\downarrow$ ”, “ $\uparrow$ ”: the disadvantage of the new algorithm over known algorithms is statistically significant; “ $\leftrightarrow$ ”, “ $\rightleftharpoons$ ”: the advantage or disadvantage is statistically insignificant. Significance level is 0.01.

For comparison, we ran local search in various GREEDY_{*r*} neighborhoods at fixed *r* value. In addition, we ran various known Variable Neighborhood Search (VNS) algorithms with GREEDY_{*r*} neighborhoods [79], see algorithms GH-VNS1–3. These algorithms use the same sequence of neighborhood types (GREEDY₁ → GREEDY_{random} → GREEDY_{*k*}) and differ in the initial neighborhood type: GREEDY₁ for GH-VNS1, GREEDY_{random} for GH-VNS2, and GREEDY_{*k*} GH-VNS3. Unlike our new AdaptiveGreedy() algorithm, GH-VNS1–3 algorithms increase *r* values, and this increase is not gradual. In addition, we included the genetic algorithm (denoted “GA-1” in Tables A1–A11) with the single-point crossover [103], real-valued genes encoded by centroid positions, and the uniform random mutation (probability 0.01). For algorithms launched in the multi-start mode (j-Means algorithm and

Lloyd’s procedure), only the best results achieved in each attempt were recorded. In Tables A1–A11, such algorithms are denoted Lloyd-MS and j-Means-MS, respectively.

The minimum, maximum, average, and median objective function values and its standard deviation were summarized after 30 runs. For all algorithms, we used the same realization of the *Lloyd()* procedure which consume the absolute majority of the computation time.

The best average and median values of the objective function (1) are underlined. We compared the new *AdaptiveGreedy()* algorithm with the known algorithm which demonstrated the best median and average results (Table 1). For comparison, we used the *t*-test [122,123] and non-parametric Wilcoxon–Mann–Whitney U test (Wilcoxon rank sum test) [124,125] with *z* approximation.

To compare the results obtained by our new algorithm, we tested the single-tailed null hypothesis $H_0: SSE_{AdaptiveGreedy} = SSE_{known}$ (the difference in the results is statistically insignificant) and the research hypothesis $H_1: SSE_{AdaptiveGreedy} < SSE_{known}$ (statistically different results, the new algorithm has an advantage). Here, $SSE_{AdaptiveGreedy}$ are results obtained by *AdaptiveGreedy()* algorithm, SSE_{known} are results of the best-known algorithm. For *t*-test comparison, we selected the algorithm lowest in average SSE value, and for Wilcoxon–Mann–Whitney U test comparison, we selected the algorithm with the lowest SSE median value. For both tests, we calculated the *p*-values (probability of the null-hypothesis acceptance), see p_t for the *t*-test and p_u for the Wilcoxon–Mann–Whitney U test in Table 1, respectively. At the selected significance level $p_{sig} = 0.01$, the null hypothesis is accepted if $p_t > 0.01$ or $p_u > 0.01$. Otherwise, the difference in algorithm results should be considered statistically significant. If the null hypothesis was accepted, we also tested a pair of single-tailed hypotheses $SSE_{AdaptiveGreedy} = SSE_{known}$ and $SSE_{AdaptiveGreedy} > SSE_{known}$.

In some cases, the Wilcoxon–Mann–Whitney test shows the statistical significance of the differences in results, while the *t*-test does not confirm the benefits of the new algorithm. Figure 5 illustrates such a situation. Both algorithms demonstrate approximately the same results. Both algorithms periodically produce results that are far from the best SSE values, which is expressed in a sufficiently large value of the standard deviation. However, the results of the new algorithm are often slightly better, which is confirmed by the rank test.

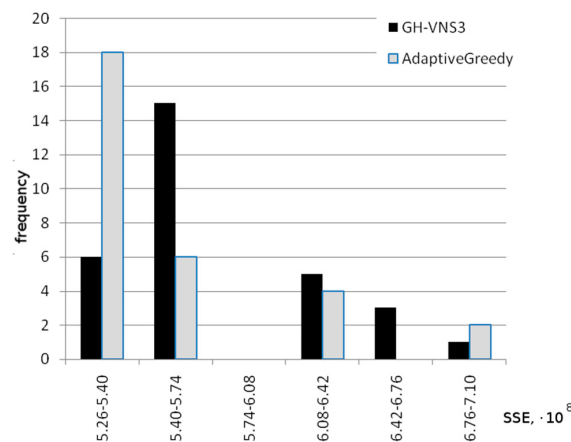


Figure 5. Frequency diagram of the results (our new algorithm vs. the best of other tested algorithms, GH-VNS3), Mopsi-Finland data set, 300 clusters, 13,467 data vectors, time limitation 5 s, 30 runs of each algorithm.

In the comparative analysis of algorithm efficiency, the choice of the unit of time plays an important role. The astronomical time spent by an algorithm strongly depends on its implementation, the ability of the compiler to optimize the program code, and the fitness of the hardware to execute the code of a specific algorithm. Algorithms are often estimated by comparing the number of iterations performed (for example, the number of population generations for a GA) or the number of evaluations of the objective function.

However, the time consumption for a single iteration of a local search algorithm depends on the neighborhood type and number of elements in the neighborhood, and this dependence can be exponential. Therefore, comparing the number of iterations is unacceptable. Comparison of the objective function calculations is also not quite correct. Firstly, the *Lloyd()* procedure which consumes almost all of the processor time, does not calculate the objective function (1) directly. Secondly, during the operation of the greedy agglomerative procedure, the number of centroids changes (decreases from $k + r$ down to k), and the time spent on computing the objective function also varies. Therefore, we nevertheless chose astronomical time as a scale for comparing algorithms. Moreover, all the algorithms use the same implementation of the *Lloyd()* algorithm launched under the same conditions.

In our computational experiments, the time limitation was used as the stop condition for all algorithms. For all data sets except the largest one, we have chosen a reasonable time limit to use the new algorithm in interactive modes. For IHEPC data and 50 clusters, a single run of the *BasicGreedy()* algorithm on the specified hardware took approximately 0.05 to 0.5 s. It is impossible to evaluate the comparative efficiency of the new algorithm in several iterations, since in this case, it does not have enough time to change the neighborhood parameter r at least once. We have increased the time to a few minutes. This time limit does not correspond to modern concepts of interactive modes of operation. Nevertheless, the rapid development of parallel computing requires the early creation of efficient algorithmic schemes. Our experiments were performed on a mass-market system. Advanced systems may cope with such large problems much faster.

As can be seen from Figure 6, the result of each algorithm depends on the elapsed time. Nevertheless, an advantage of the new algorithm is evident regardless of the chosen time limit.

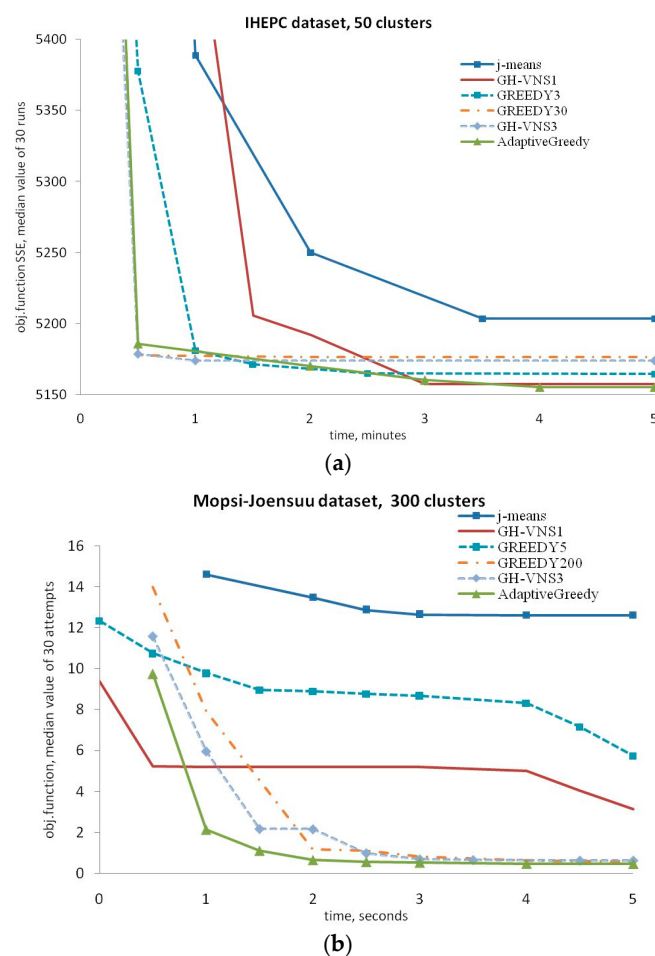


Figure 6. Cont.

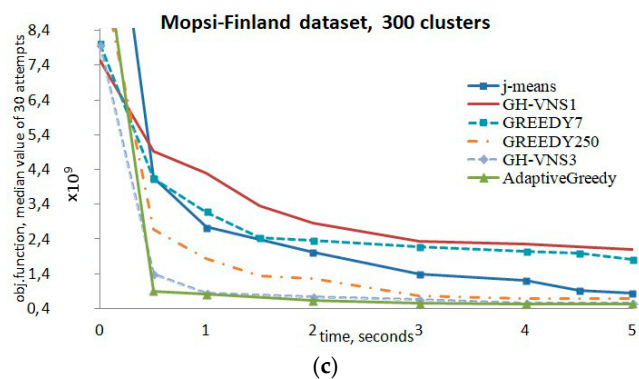


Figure 6. Comparative analysis of the convergence speed. Dependence of the median result on computation time for: (a) Individual Household Electric Power Consumption (IHEPC) data set, 50 clusters, 2,075,259 data vectors, time limitation 5 min; (b) Mopsi-Joensuu data set, 300 clusters, 6014 data vectors, time limitation 5 s; (c) Mopsi-Finland data set, 300 clusters, 13,467 data vectors, time limitation 5 s.

To test the scalability of the proposed approach and the efficiency of the new algorithm on other hardware, we carried out additional experiments with NVIDIA GeForce 9600GT GPU, 2048 MB RAM, 336 GFLOPS. The declared performance of this simpler equipment is approximately 6 times lower. The results of experiments with proportional increase of time limitation are shown in Table 2. The difference with the results in Table 1 is obviously insignificant.

Table 2. Additional benchmarking on NVIDIA GeForce 9600GT GPU. Comparative results for Mopsi-Finland data set. 13,467 data vectors in $\mathbb{R}^2$, time limitation 30 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
$k = 300$					
GH-VNS3	5.33373×10^8	7.29800×10^8	5.85377×10^8	5.52320×10^8	5.59987×10^7
AdaptiveGreedy	5.27254×10^8	7.09410×10^8	5.59033×10^8	5.38888×10^8	4.60585×10^7
$k = 30$					
GH-VNS2	3.42528×10^{10}	3.48723×10^{10}	3.43916×10^{10}	3.43474×10^{10}	1.46818×10^8
GH-VNS3	3.42528×10^{10}	3.46408×10^{10}	3.43731×10^{10}	3.43474×10^{10}	7.81989×10^7
AdaptiveGreedy	3.42528×10^{10}	3.46274×10^{10}	3.43337×10^{10}	3.43473×10^{10}	8.13882×10^7

The ranges of SSE values in the majority of Tables A1–A11 are narrow, nevertheless, the differences are statistically significant in several cases, see Table 1. In all cases, our new algorithm outperforms known ones or demonstrates approximately the same efficiency (difference in the results is statistically insignificant). Moreover, the new algorithm demonstrates the stability of its results (narrow range of objective function values).

Search results in both $SWAP_r$ and $GREEDY_r$ neighborhoods depend on a correct choice of parameter r (the number of replaced or added centroids). However, in general, local search algorithms with $GREEDY_r$ neighborhoods outperform the $SWAP_r$ neighborhood search. A simple reconnaissance search procedure enables the further improvement of the efficiency.

4. Discussion

The advantages of our algorithm are statistically significant for a large problem (IHEPC data), as well as for problems with a complex data structure (Mopsi-Joensuu and Mopsi-Finland data). The Mopsi data sets contain geographic coordinates of Mopsi users, which are extremely unevenly distributed in accordance with the natural organization of the urban environment, depending on street directions and urban infrastructure (Figure 6). In this case, the aim of clustering is to find some natural

groups of users according to a geometric/geographic principle for assigning them to k service centers (hubs) such as shopping centers, bus stops, wireless network base stations, etc.

Often, geographical data sets show such a disadvantage of Lloyd's procedure as its inability to find a solution close to the exact one. Often, on such data, the value of the objective function found by the Lloyd's procedure in the multi-start mode turns out to be many times greater than the values obtained by other algorithms, such as J-Means or RVNS algorithms with SWAP neighborhoods. As can be seen from Tables A2, A3 and A5 in Appendix A, for such data, GREEDY_r neighborhoods search provides significant advantages within a limited time, and our new self-adjusting *AdaptiveGreedy()* solver enhances these advantages.

The VNS algorithmic framework is useful for creating effective computational tools intended to solve complex practical problems. Embedding the most efficient types of neighborhoods in this framework depends on the problem type being solved. In problems such as k-means, the search in neighborhoods with specific parameters strongly depends not only on the generalized numerical parameters of the problems, such as the number of clusters, number of data vectors, and the search space dimensionality, but also on the internal data structure. In general, the comparative efficiency of the search in GREEDY_r neighborhoods for certain types of practical problems and for specific data sets remains an open question. Nevertheless, the algorithm presented in this work, which automatically performs the adjustment of the most important parameter of such neighborhoods, enables its user to obtain the best result which the variable neighborhood search in GREEDY_r is able to provide, without preliminary experiments in all possible GREEDY_r neighborhoods. Thus, the new algorithm is a more versatile computational tool in comparison with the known VNS algorithms.

Greedy agglomerative procedures are widely used as crossover operators in genetic algorithms [46,88,90,110]. In this case, most often, the "parent" solutions are merged completely to obtain an intermediate solution with an excessive number of centers or centroids [46,88], which corresponds to the search in the GREEDY_k neighborhood (one of the crossed "parent" solutions acts as the parameter S_2), although, other versions of the greedy agglomerative crossover operator are also possible [90,110]. Such algorithms successfully compete with the advanced local search algorithms discussed in this article.

Self-configuring evolutionary algorithms [126–128] have been widely used for solving various optimization problems. An important direction of the further research is to study the possibility of adjusting the parameter r in greedy agglomerative crossover operators of genetic algorithms. Such procedures with self-adjusting parameter r could lead to a further increase in the accuracy of solving the k -means problem with respect to the achieved value of the objective function. Such evolutionary algorithms could also involve a reconnaissance search, which would then continue by applying the greedy agglomerative crossover operator with r values chosen from the most favorable range.

In addition, the similarity in problem statements of the k -means, k -medoids and k -median problems promises us a reasonable hope for the applicability of the same approaches to improving the accuracy of algorithms, including VNS algorithms, by adjusting the parameter r of the neighborhoods similar with GREEDY_r.

5. Conclusions

The process of introducing machine learning methods into all spheres of life determines the need to develop not only fast, but also the most accurate algorithms for solving related optimization problems. As practice shows, including this study, when solving some problems, the most popular clustering algorithm gives a result extremely far from the optimal k -means problem solution.

In this research, we introduced GREEDY_r search neighborhoods and found that searching in both SWAP and GREEDY_r neighborhoods has advantages over the simplest Lloyd's procedure. However, the results strongly depend on the parameters of such neighborhoods, and the optimal values of these parameters differ significantly for test problems. Nevertheless, searching in GREEDY_r neighborhoods outperforms searching in SWAP neighborhoods in terms of accuracy.

We hope that our new variable neighborhood search algorithm (solver) for GPUs, which is more versatile due to its self-adjusting capability and has an advantage with respect to the accuracy of solving the k-means problem over known algorithms, will encourage researchers and practitioners in the field of machine learning to build competitive systems with the lowest possible error within a limited time. Such systems should be in demand when clustering geographic data, as well as when solving a wide range of problems with the highest cost of error.

Author Contributions: Conceptualization, L.K. and I.R.; methodology, L.K.; software, L.K.; validation, I.R. and E.T.; formal analysis, I.R. and A.P.; investigation, I.R.; resources, L.K. and E.T.; data curation, I.R.; writing—original draft preparation, L.K. and I.R.; writing—review and editing, L.K., E.T., and A.P.; visualization, I.R.; supervision, L.K.; project administration, L.K.; funding acquisition, L.K. and A.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by The Ministry of Science and Higher Education of the Russian Federation, project No. FEFE-2020-0013.

Conflicts of Interest: The authors declare no conflict of interest. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

NP	Non-deterministic polynomial-time
MSSC	Minimum Sum-of-Squares Clustering
SSE	Sum of Squared Errors
ALA algorithm	Alternate Location-Allocation algorithm
VNS	Variable Neighborhood Search
GA	Genetic Algorithm
IBC	Information Bottleneck Clustering
VND	Variable Neighborhood Descent
RVNS	Randomized Variable Neighborhood Search
GPU	Graphics Processing Unit
CPU	Central Processing Unit
RAM	Random Access Memory
CUDA	Compute Unified Device Architecture
IHEPC	Individual Household Electric Power Consumption
Lloyd-MS	Lloyd's procedure in a multi-start mode
J-means-MS	J-Means algorithm in a multi-start mode (SWAP ₁ +Lloyd VND)
GREEDY _r	A neighborhood formed by applying greedy agglomerative procedures with <i>r</i> excessive clusters, and the RVNS algorithm which combines search in such neighborhood with Lloyd's procedure
SWAP _r	A neighborhood formed by replacing <i>r</i> centroids by data vectors, and the RVNS algorithm which combines search in such neighborhood with Lloyd's procedure
GH-VNS1	VNS algorithm with GREEDY _r neighborhoods and GREEDY ₁ for the initial neighborhood type
GH-VNS2	VNS algorithm with GREEDY _r neighborhoods and GREEDY _{random} for the initial neighborhood type
GH-VNS3	VNS algorithm with GREEDY _r neighborhoods and GREEDY _k for the initial neighborhood type
GA-1	Genetic algorithm with the single-point crossover, real-valued genes encoded by centroid positions, and the uniform random mutation
AdaptiveGreedy	New algorithm proposed in this article

Appendix A. Results of Computational Experiments

Table A1. Comparative results for Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 30$ clusters, time limitation 5 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	35.5712	43.3993	39.1185	38.7718	2.9733
j-Means-MS	18.4076	23.7032	20.3399	19.8533	1.8603
GREEDY ₁	18.3253	27.6990	21.4555	21.6629	3.1291
GREEDY ₂	18.3253	21.7008	19.3776	18.3254	1.6119
GREEDY ₃	18.3145	21.7007	18.5817	18.3254	0.9372
GREEDY ₅	18.3253	21.7007	18.5129	18.3254	0.7956
GREEDY ₇	18.3253	21.7008	18.5665	18.3255	0.9021
GREEDY ₁₀	18.3253	21.7010	18.5666	18.3255	0.9021
GREEDY ₁₂	18.3254	21.7009	18.5852	18.3256	0.9362
GREEDY ₁₅	18.3254	18.3257	18.3255	18.3255	0.0001
GREEDY ₂₀	18.3254	18.3263	18.3257	18.3257	0.0002
GREEDY ₂₅	18.3254	18.3257	18.3255	18.3255	0.0001
GREEDY ₃₀	18.3254	18.3261	18.3258	18.3258	0.0002
GH-VNS1	18.3147	18.3255	18.3238	18.3253	0.0039
GH-VNS2	18.3253	21.7008	19.3776	18.3254	1.6119
GH-VNS3	18.3146	21.6801	18.5634	18.3254	0.8971
SWAP ₁ (the best of SWAP _r)	18.9082	20.3330	19.4087	18.9967	0.6019
GA-1	18.6478	21.1531	19.9555	19.9877	0.6632
AdaptiveGreedy	18.3146	18.3258	18.3240	18.3253	0.0037

Table A2. Comparative results for Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 100$ clusters, time limitation 5 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	23.1641	34.7834	27.5520	27.1383	3.6436
j-Means-MS	1.7628	31.8962	11.1832	2.4216	11.7961
GREEDY ₁	20.6701	35.5447	28.9970	29.2429	5.0432
GREEDY ₂	2.8264	29.0682	9.9708	5.3363	9.6186
GREEDY ₃	2.6690	10.5998	4.1444	3.0588	2.2108
GREEDY ₅	1.9611	4.3128	2.7385	2.7299	0.6135
GREEDY ₇	2.0837	4.6443	2.8730	2.6358	0.7431
GREEDY ₁₀	1.9778	3.8635	2.5613	2.3304	0.6126
GREEDY ₁₂	1.7817	4.3023	2.5639	2.2009	0.8730
GREEDY ₁₅	1.9564	3.1567	2.3884	2.2441	0.3620
GREEDY ₂₀	1.7937	3.2809	2.4542	2.3500	0.4746
GREEDY ₂₅	1.9532	3.3874	2.4195	2.2575	0.5470
GREEDY ₃₀	1.9274	2.4580	2.1723	2.1458	0.2171
GREEDY ₅₀	1.8903	9.3675	2.8047	2.1614	2.0838
GREEDY ₇₅	1.7878	2.8855	2.1775	2.0272	0.4023
GREEDY ₁₀₀	1.8021	2.2942	2.0158	1.9849	0.1860
GH-VNS1	2.8763	17.1139	7.3196	4.3341	5.7333
GH-VNS2	2.8264	29.0682	9.9708	5.3363	9.6186
GH-VNS3	1.7643	2.7357	2.0513	1.9822	0.2699
SWAP ₃ (the best of rand. SWAP _r)	4.9739	23.6572	9.0159	8.3907	4.1351
GA-1	4.8922	19.1543	8.5914	7.1764	4.1096
AdaptiveGreedy	1.7759	2.3265	1.9578	1.9229	0.1523

Table A3. Comparative results for Mopsi-Joensuu data set. 6014 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 5 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	4.1789	14.7570	9.1143	9.3119	3.0822
j-Means-MS	7.0119	22.3126	14.2774	12.6199	5.5095
GREEDY ₁	7.1654	15.3500	9.6113	9.2176	2.5266
GREEDY ₂	4.9896	14.4839	8.9197	8.2013	3.3072
GREEDY ₃	5.8967	14.1110	8.3260	8.0441	2.2140
GREEDY ₅	2.9115	10.2536	5.8012	5.7305	2.2740
GREEDY ₇	2.6045	7.9868	4.4201	4.0548	1.4841
GREEDY ₁₀	2.5497	8.6758	4.1796	2.9639	1.8494
GREEDY ₁₂	2.0753	4.7134	3.0383	2.8777	0.8348
GREEDY ₁₅	1.8975	8.7890	3.8615	3.2661	1.8064
GREEDY ₂₀	1.1878	3.7944	2.4577	2.4882	0.9554
GREEDY ₂₅	1.1691	3.5299	1.8489	1.6407	0.7460
GREEDY ₃₀	1.1151	4.9425	2.3711	2.0582	1.1501
GREEDY ₅₀	1.3526	3.5471	1.8635	1.7114	0.6046
GREEDY ₇₅	1.0533	5.5915	1.9129	1.4261	1.2082
GREEDY ₁₀₀	0.8047	2.0349	1.2602	1.1994	0.3811
GREEDY ₁₅₀	0.6243	1.4755	0.8743	0.8301	0.2447
GREEDY ₂₀₀	0.4555	1.0154	0.6746	0.5882	0.2103
GREEDY ₂₅₀	0.4789	1.3368	0.7233	0.6695	0.2164
GREEDY ₃₀₀	0.5474	1.0472	0.7228	0.6657	0.1419
GH-VNS1	1.6219	5.2528	3.0423	3.1332	1.0222
GH-VNS2	1.2073	8.6144	3.2228	2.3501	2.4014
GH-VNS3	0.4321	0.6838	0.6024	0.6139	0.0836
SWAP ₁₂ (the best of SWAP by median)	2.6016	5.5038	3.6219	3.3612	1.0115
SWAP ₂₀ (the best of SWAP by avg.)	2.1630	5.1235	3.4958	3.4076	0.8652
GA-1	5.4911	12.6950	8.8799	7.7181	2.5384
AdaptiveGreedy	0.3128	0.6352	0.4672	0.4604	0.1026

Table A4. Comparative results for Mopsi-Finland data set. 13,467 data vectors in $\mathbb{R}^2$, $k = 30$ clusters, time limitation 5 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	4.79217×10^{10}	6.36078×10^{10}	5.74896×10^{10}	5.79836×10^{10}	3.69760×10^9
j-Means-MS	3.43535×10^{10}	4.26830×10^{10}	3.66069×10^{10}	3.60666×10^{10}	1.75725×10^9
GREEDY ₁	3.43195×10^{10}	3.70609×10^{10}	3.51052×10^{10}	3.48431×10^{10}	7.42636×10^8
GREEDY ₂	3.43194×10^{10}	3.49405×10^{10}	3.44496×10^{10}	3.44140×10^{10}	1.64360×10^8
GREEDY ₃	3.43195×10^{10}	3.49411×10^{10}	3.44474×10^{10}	3.44140×10^{10}	1.71131×10^8
GREEDY ₅	3.43195×10^{10}	3.48411×10^{10}	3.44663×10^{10}	3.44141×10^{10}	1.65153×10^8
GREEDY ₇	3.42531×10^{10}	3.47610×10^{10}	3.44091×10^{10}	3.43504×10^{10}	1.76023×10^8
GREEDY ₁₀	3.42560×10^{10}	3.48824×10^{10}	3.45106×10^{10}	3.43573×10^{10}	2.36526×10^8
GREEDY ₁₂	3.42606×10^{10}	3.48822×10^{10}	3.44507×10^{10}	3.43901×10^{10}	1.68986×10^8
GREEDY ₁₅	3.42931×10^{10}	3.47817×10^{10}	3.43874×10^{10}	3.43901×10^{10}	8.31510×10^7
GREEDY ₂₀	3.42954×10^{10}	3.48826×10^{10}	3.44186×10^{10}	3.43905×10^{10}	1.28972×10^8
GREEDY ₂₅	3.43877×10^{10}	3.44951×10^{10}	3.43982×10^{10}	3.43907×10^{10}	2.57320×10^7
GREEDY ₃₀	3.43900×10^{10}	3.48967×10^{10}	3.45169×10^{10}	3.43979×10^{10}	1.93565×10^8
GH-VNS1	3.42626×10^{10}	3.48724×10^{10}	3.45244×10^{10}	3.44144×10^{10}	2.00510×10^8
GH-VNS2	3.42528×10^{10}	3.48723×10^{10}	3.44086×10^{10}	3.43474×10^{10}	1.54771×10^8
GH-VNS3	3.42528×10^{10}	3.47955×10^{10}	3.43826×10^{10}	3.43474×10^{10}	1.02356×10^8
SWAP ₁ (the best of SWAP _r)	3.43199×10^{10}	3.55777×10^{10}	3.46821×10^{10}	3.46056×10^{10}	3.22711×10^8
GA-1	3.48343×10^{10}	3.81846×10^{10}	3.65004×10^{10}	3.64415×10^{10}	1.00523×10^9
AdaptiveGreedy	3.42528×10^{10}	3.47353×10^{10}	3.43385×10^{10}	3.43473×10^{10}	1.03984×10^8

Table A5. Comparative results for Mopsi- Finland data set. 13,467 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 5 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	5.41643×10^9	6.89261×10^9	6.25619×10^9	6.24387×10^9	3.23827×10^8
j-Means-MS	6.75216×10^8	1.38889×10^9	8.92782×10^8	8.35397×10^8	1.86995×10^8
GREEDY ₁	4.08445×10^9	9.07208×10^9	5.89974×10^9	5.59903×10^9	1.47601×10^8
GREEDY ₂	1.11352×10^9	2.10247×10^9	1.59229×10^9	1.69165×10^9	2.89625×10^8
GREEDY ₃	9.63842×10^8	2.15674×10^9	1.61490×10^9	1.60123×10^9	3.06567×10^8
GREEDY ₅	9.11944×10^8	2.36799×10^9	1.66021×10^9	1.70448×10^9	3.68575×10^8
GREEDY ₇	1.17328×10^9	2.44476×10^9	1.77589×10^9	1.80948×10^9	2.68354×10^8
GREEDY ₁₀	1.14221×10^9	2.00426×10^9	1.67586×10^9	1.69601×10^9	2.14822×10^8
GREEDY ₁₂	9.41133×10^8	2.28940×10^9	1.59715×10^9	1.62288×10^9	3.01841×10^8
GREEDY ₁₅	8.86983×10^8	2.29776×10^9	1.53989×10^9	1.43319×10^9	3.70138×10^8
GREEDY ₂₀	1.02224×10^9	2.11636×10^9	1.62601×10^9	1.64029×10^9	2.45576×10^8
GREEDY ₂₅	9.07984×10^8	1.87134×10^9	1.42878×10^9	1.42864×10^9	2.74744×10^8
GREEDY ₃₀	8.44247×10^8	2.22882×10^9	1.50817×10^9	1.56015×10^9	3.52497×10^8
GREEDY ₅₀	7.98191×10^8	1.68198×10^9	1.26851×10^9	1.17794×10^9	2.67082×10^8
GREEDY ₇₅	6.97650×10^8	1.74139×10^9	1.16422×10^9	1.16616×10^9	2.82454×10^8
GREEDY ₁₀₀	6.55465×10^8	1.44162×10^9	1.03643×10^9	1.09001×10^9	1.95246×10^8
GREEDY ₁₅₀	5.94256×10^8	1.45317×10^9	8.88898×10^8	7.96787×10^8	2.33137×10^8
GREEDY ₂₀₀	5.60885×10^8	1.41411×10^9	7.96908×10^8	7.20282×10^8	2.26191×10^8
GREEDY ₂₅₀	5.58602×10^8	1.13946×10^9	7.58434×10^8	6.81196×10^8	1.65511×10^8
GREEDY ₃₀₀	5.68646×10^8	1.41338×10^9	7.35067×10^8	6.83004×10^8	1.76126×10^8
GH-VNS1	1.40141×10^9	2.86919×10^9	2.16238×10^9	2.10817×10^9	3.42105×10^8
GH-VNS2	8.22679×10^8	2.12228×10^9	1.40322×10^9	1.39457×10^9	2.96599×10^8
GH-VNS3	<u>5.33373×10^8</u>	7.29800×10^8	<u>5.74914×10^8</u>	<u>5.48427×10^8</u>	<u>5.05346×10^7</u>
SWAP ₁ (the best of SWAP _r)	6.69501×10^8	9.06507×10^8	7.48932×10^8	7.35532×10^8	6.74846×10^7
GA-1	4.54419×10^9	7.11460×10^9	5.67688×10^9	5.61135×10^9	5.99687×10^8
AdaptiveGreedy	<u>5.27254×10^8</u>	7.09410×10^8	<u>5.60867×10^8</u>	<u>5.38952×10^8</u>	<u>4.89257×10^7</u>

Table A6. Comparative results for BIRCH3 data set. 10^5 data vectors in $\mathbb{R}^2$, $k = 100$ clusters, time limitation 10 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	8.13022×10^{13}	9.51129×10^{13}	8.96327×10^{13}	9.06147×10^{13}	4.84194×10^{12}
j-Means-MS	4.14627×10^{13}	6.25398×10^{13}	4.78063×10^{13}	4.55711×10^{13}	6.89734×10^{12}
GREEDY ₁	3.73299×10^{13}	5.64559×10^{13}	4.13352×10^{13}	3.90845×10^{13}	5.19021×10^{12}
GREEDY ₂	3.71499×10^{13}	3.72063×10^{13}	3.71689×10^{13}	3.71565×10^{13}	2.44802×10^{10}
GREEDY ₃	3.71518×10^{13}	3.72643×10^{13}	3.71840×10^{13}	3.71545×10^{13}	4.12818×10^{10}
GREEDY ₅	3.71485×10^{13}	3.72087×10^{13}	<u>3.71644×10^{13}</u>	<u>3.71518×10^{13}</u>	2.22600×10^{10}
GREEDY ₇	3.71518×10^{13}	3.72267×10^{13}	3.71755×10^{13}	3.71658×10^{13}	2.24845×10^{10}
GREEDY ₁₀	3.71555×10^{13}	3.72119×10^{13}	3.71771×10^{13}	3.71794×10^{13}	1.90289×10^{10}
GREEDY ₁₂	3.71556×10^{13}	3.72954×10^{13}	3.71892×10^{13}	3.71693×10^{13}	3.91673×10^{10}
GREEDY ₁₅	3.71626×10^{13}	3.72169×10^{13}	3.71931×10^{13}	3.71963×10^{13}	<u>1.86102×10^{10}</u>
GREEDY ₂₀	3.71600×10^{13}	3.72638×10^{13}	3.72118×10^{13}	3.72153×10^{13}	2.69206×10^{10}
GREEDY ₂₅	3.72042×10^{13}	3.72690×10^{13}	3.72284×10^{13}	3.72228×10^{13}	2.14437×10^{10}
GREEDY ₃₀	3.72180×10^{13}	3.73554×10^{13}	3.72586×10^{13}	3.72471×10^{13}	4.33818×10^{10}
GREEDY ₅₀	3.72166×10^{13}	3.76422×10^{13}	3.73883×10^{13}	3.73681×10^{13}	16.1061×10^{10}
GREEDY ₇₅	3.72399×10^{13}	3.84870×10^{13}	3.76286×10^{13}	3.74750×10^{13}	41.6632×10^{10}
GREEDY ₁₀₀	3.72530×10^{13}	3.91589×10^{13}	3.80730×10^{13}	3.84482×10^{13}	61.9706×10^{10}
GH-VNS1	3.71914×10^{13}	3.77527×10^{13}	3.73186×10^{13}	3.72562×10^{13}	18.3590×10^{10}
GH-VNS2	3.71568×10^{13}	3.73791×10^{13}	3.72116×10^{13}	3.72051×10^{13}	6.08081×10^{10}
GH-VNS3	3.71619×10^{13}	3.73487×10^{13}	3.72387×10^{13}	3.72282×10^{13}	5.96618×10^{10}
SWAP ₁ (the best of SWAP _r)	4.28705×10^{13}	5.48014×10^{13}	4.82383×10^{13}	4.75120×10^{13}	3.90128×10^{12}
GA-1	3.84317×10^{13}	4.08357×10^{13}	3.97821×10^{13}	3.97088×10^{13}	7.43642×10^{11}
AdaptiveGreedy	<u>3.71484×10^{13}</u>	3.72011×10^{13}	3.71726×10^{13}	3.71749×10^{13}	2.02784×10^{10}

Table A7. Comparative results for BIRCH3 data set. 10^5 data vectors in $\mathbb{R}^2$, $k = 300$ clusters, time limitation 10 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	3.49605×10^{13}	4.10899×10^{13}	3.74773×10^{13}	3.77191×10^{13}	2.32012×10^{12}
j-Means-MS	1.58234×10^{13}	2.02926×10^{13}	1.75530×10^{13}	1.70507×10^{13}	1.43885×10^{12}
GREEDY ₁	1.48735×10^{13}	2.63695×10^{13}	1.71372×10^{13}	1.60354×10^{13}	2.98555×10^{12}
GREEDY ₂	1.31247×10^{13}	1.45481×10^{13}	1.37228×10^{13}	1.36745×10^{13}	4.01697×10^{11}
GREEDY ₃	1.34995×10^{13}	1.49226×10^{13}	1.39925×10^{13}	1.39752×10^{13}	4.85917×10^{11}
GREEDY ₅	1.33072×10^{13}	1.45757×10^{13}	1.39069×10^{13}	1.38264×10^{13}	4.46890×10^{11}
GREEDY ₇	1.34959×10^{13}	1.49669×10^{13}	1.41606×10^{13}	1.41764×10^{13}	4.92200×10^{11}
GREEDY ₁₀	1.31295×10^{13}	1.42722×10^{13}	1.35970×10^{13}	1.35318×10^{13}	3.70511×10^{11}
GREEDY ₁₂	1.32677×10^{13}	1.49028×10^{13}	1.35561×10^{13}	1.33940×10^{13}	4.44283×10^{11}
GREEDY ₁₅	1.32077×10^{13}	1.41079×10^{13}	1.34102×10^{13}	1.33832×10^{13}	2.16247×10^{11}
GREEDY ₂₀	1.31994×10^{13}	1.43160×10^{13}	1.35420×10^{13}	1.34096×10^{13}	3.43684×10^{11}
GREEDY ₂₅	1.31078×10^{13}	1.37699×10^{13}	1.33571×10^{13}	1.33040×10^{13}	2.16378×10^{11}
GREEDY ₃₀	1.32947×10^{13}	1.45967×10^{13}	1.37618×10^{13}	1.36729×10^{13}	3.92767×10^{11}
GREEDY ₅₀	1.32284×10^{13}	1.38691×10^{13}	1.34840×10^{13}	1.33345×10^{13}	2.70770×10^{11}
GREEDY ₇₅	1.30808×10^{13}	1.33266×10^{13}	1.31857×10^{13}	1.31833×10^{13}	7.22941×10^{10}
GREEDY ₁₀₀	1.30852×10^{13}	1.32697×10^{13}	1.31250×10^{13}	1.31067×10^{13}	4.94315×10^{10}
GREEDY ₁₅₀	1.30754×10^{13}	1.31446×10^{13}	1.30971×10^{13}	1.30952×10^{13}	1.82873×10^{10}
GREEDY ₂₀₀	1.30773×10^{13}	1.31172×10^{13}	1.30916×10^{13}	1.30912×10^{13}	1.08001×10^{10}
GREEDY ₂₅₀	1.30699×10^{13}	1.31073×10^{13}	1.30944×10^{13}	1.30990×10^{13}	1.18367×10^{10}
GREEDY ₃₀₀	1.30684×10^{13}	1.31068×10^{13}	1.30917×10^{13}	1.30933×10^{13}	1.21748×10^{10}
GH-VNS1	1.40452×10^{13}	1.56256×10^{13}	1.45212×10^{13}	1.42545×10^{13}	55.7231×10^{10}
GH-VNS2	1.32287×10^{13}	1.38727×10^{13}	1.34654×10^{13}	1.34568×10^{13}	2.01065×10^{11}
GH-VNS3	1.30996×10^{13}	1.31378×10^{13}	1.31158×10^{13}	1.31138×10^{13}	1.44998×10^{10}
SWAP ₂ (the best of SWAP _r by median)	2.18532×10^{13}	3.25705×10^{13}	2.54268×10^{13}	2.37312×10^{13}	3.78491×10^{12}
SWAP ₇ (the best of SWAP _r by avg.)	2.24957×10^{13}	2.86883×10^{13}	2.46775×10^{13}	2.47301×10^{13}	1.51198×10^{12}
GA-1	1.38160×10^{13}	1.71472×10^{13}	1.55644×10^{13}	1.54336×10^{13}	9.21217×10^{11}
AdaptiveGreedy	1.30807×10^{13}	1.31113×10^{13}	1.30922×10^{13}	1.30925×10^{13}	0.87731×10^{10}

Table A8. Comparative results for S1 data set. 5000 data vectors in $\mathbb{R}^2$, $k = 15$ clusters, time limitation 1 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	8.91703×10^{12}	8.91707×10^{12}	8.91704×10^{12}	8.91703×10^{12}	1.31098×10^7
j-Means-MS	8.91703×10^{12}	14.2907×10^{12}	12.1154×10^{12}	13.3667×10^{12}	2.38947×10^{12}
GREEDY ₁	8.91703×10^{12}	13.2502×10^{12}	9.27814×10^{12}	8.91703×10^{12}	1.25086×10^{12}
GREEDY ₂	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.00000
GREEDY ₃	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.00000
GREEDY ₅	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	4.03023×10^5
GREEDY ₇	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	4.87232×10^5
GREEDY ₁₀	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	5.12234×10^5
GREEDY ₁₂	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	3.16158×10^5
GREEDY ₁₅	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	5.01968×10^5
GH-VNS1	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.00000
GH-VNS2	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.00000
GH-VNS3	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	4.03023×10^5
SWAP ₁ (the best of SWAP)	8.91703×10^{12}	8.91709×10^{12}	8.91704×10^{12}	8.91703×10^{12}	8.67594×10^6
GA-1	8.91703×10^{12}	8.91707×10^{12}	8.91703×10^{12}	8.91703×10^{12}	9.04519×10^6
AdaptiveGreedy	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	8.91703×10^{12}	0.00000

Table A9. Comparative results for S1 data set. 5000 data vectors in $\mathbb{R}^2$, $k = 50$ clusters, time limitation 1 s.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	3.94212×10^{12}	4.06133×10^{12}	3.99806×10^{12}	3.99730×10^{12}	4.52976×10^{10}
j-Means-MS	3.96626×10^{12}	4.40078×10^{12}	4.12311×10^{12}	4.07123×10^{12}	14.81090×10^{10}
GREEDY ₁	3.82369×10^{12}	4.19102×10^{12}	3.91601×10^{12}	3.88108×10^{12}	9.82433×10^{10}
GREEDY ₂	3.74350×10^{12}	3.76202×10^{12}	3.75014×10^{12}	3.74936×10^{12}	6.10139×10^9
GREEDY ₃	3.74776×10^{12}	3.76237×10^{12}	3.75455×10^{12}	3.75456×10^{12}	5.24513×10^9
GREEDY ₅	3.74390×10^{12}	3.77031×10^{12}	3.75345×10^{12}	3.75298×10^{12}	7.17733×10^9
GREEDY ₇	3.74446×10^{12}	3.77208×10^{12}	3.75277×10^{12}	3.75190×10^{12}	7.40052×10^9
GREEDY ₁₀	3.74493×10^{12}	3.76031×10^{12}	3.75159×10^{12}	3.75185×10^{12}	5.26553×10^9
GREEDY ₁₅	3.74472×10^{12}	3.77922×10^{12}	3.75426×10^{12}	3.75519×10^{12}	9.79855×10^9
GREEDY ₂₀	3.75028×10^{12}	3.76448×10^{12}	3.75586×10^{12}	3.75573×10^{12}	3.97310×10^9
GREEDY ₂₅	3.74770×10^{12}	3.76224×10^{12}	3.75500×10^{12}	3.75572×10^{12}	4.95370×10^9
GREEDY ₃₀	3.75014×10^{12}	3.76010×10^{12}	3.75583×10^{12}	3.75661×10^{12}	3.45280×10^9
GREEDY ₅₀	3.74676×10^{12}	3.77396×10^{12}	3.76021×10^{12}	3.75933×10^{12}	9.09159×10^9
GH-VNS1	3.74310×10^{12}	3.76674×10^{12}	3.74911×10^{12}	3.74580×10^{12}	6.99859×10^9
GH-VNS2	3.75106×10^{12}	3.77369×10^{12}	3.75792×10^{12}	3.75782×10^{12}	6.67960×10^9
GH-VNS3	3.75923×10^{12}	3.77964×10^{12}	3.76722×10^{12}	3.76812×10^{12}	6.00125×10^9
SWAP ₃ (the best of SWAP)	3.75128×10^{12}	3.79170×10^{12}	3.77853×10^{12}	3.77214×10^{12}	4.53608×10^9
GA-1	3.84979×10^{12}	3.99291×10^{12}	3.92266×10^{12}	3.92818×10^{12}	4.56845×10^{12}
AdaptiveGreedy	3.74340×10^{12}	3.76313×10^{12}	3.74851×10^{12}	3.75037×10^{12}	5.56298×10^9

Table A10. Comparative results for Individual Household Electric Power Consumption (IHEPC) data set. 2,075,259 data vectors in $\mathbb{R}^7$, $k = 15$ clusters, time limitation 5 min.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	AVERAGE	Median	Std.dev
Lloyd-MS	12,874.8652	12,880.0703	12,876.0219	12,874.8652	2.2952
j-Means-MS	12,874.8652	13,118.6455	12,984.7081	12,962.1323	75.6539
all GREEDY ₁₋₁₅ (equal results)	12,874.8633	12,874.8633	12,874.8633	12,874.8633	0.0000
GH-VNS1	12,874.8633	12,874.8633	12,874.8633	12,874.8633	0.0000
GH-VNS2	12,874.8633	12,874.8633	12,874.8633	12,874.8633	0.0000
GH-VNS3	12,874.8633	12,874.8633	12,874.8633	12,874.8633	0.0000
GA-1	12,874.8643	12,874.8652	12,874.8644	12,874.8643	0.0004
AdaptiveGreedy	12,874.8633	12,874.8633	12,874.8633	12,874.8633	0.0000

Table A11. Comparative results for Individual Household Electric Power Consumption (IHEPC) data set. 2,075,259 data vectors in $\mathbb{R}^7$, $k = 50$ clusters, time limitation 5 min.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
Lloyd-MS	5605.0625	5751.1982	5671.0820	5660.4429	54.2467
j-Means-MS	5160.2700	6280.6440	5496.6539	5203.5679	493.7311
GREEDY ₁	5200.9268	5431.3647	5287.4101	5281.7300	77.0460
GREEDY ₂	5167.1482	5283.3894	5171.6509	5192.1274	7.7203
GREEDY ₃	5155.5166	5178.4063	5166.5360	5164.6045	8.1580
GREEDY ₅	5164.6040	5178.4336	5170.8829	5174.0938	6.0904
GREEDY ₇	5162.5381	5178.1269	5168.7218	5171.8292	6.4518
GREEDY ₁₀	5154.2017	5176.4502	5162.0460	5160.4014	7.2029
GREEDY ₁₂	5162.8715	5181.0281	5166.8952	5165.3295	6.0172
GREEDY ₁₅	5163.2500	5181.1333	5167.3385	5165.8037	5.7910
GREEDY ₂₀	5156.2852	5176.6855	5166.2013	5164.6323	7.8749

Table A11. Cont.

Algorithm or Neighborhood	Achieved SSE Summarized After 30 Runs				
	Min (Record)	Max (Worst)	Average	Median	Std.dev
GREEDY ₂₅	5166.9820	5181.8529	5175.0317	5176.2136	6.1471
GREEDY ₃₀	5168.6309	5182.4351	5175.2414	5176.4512	6.4635
GREEDY ₅₀	5168.3887	5182.4321	5177.5249	5177.6855	5.4437
GH-VNS1	5155.5166	5164.6313	5158.6549	5157.6812	3.7467
GH-VNS2	5159.8818	5176.6855	5167.3365	5166.9512	5.6808
GH-VNS3	5171.2969	5182.4321	5175.0468	5174.0752	3.6942
GA-1	5215.9521	5248.4521	5230.2839	5226.0386	13.2694
AdaptiveGreedy	5153.5640	5163.9316	5157.0822	5155.5198	3.6034

References

- Berkhin, P. *Survey of Clustering Data Mining Techniques*; Accrue Software: New York, NY, USA, 2002.
- Cormack, R.M. A Review of Classification. *J. R. Stat. Soc. Ser. A* **1971**, *134*, 321–367. [\[CrossRef\]](#)
- Tsai, C.Y.; Chiu, C.C. A VNS-based hierarchical clustering method. In Proceedings of the 5th WSEAS International Conference on Computational Intelligence, Man-Machine Systems and Cybernetics (CIMMACS'06), Venice, Italy, 20–22 November 2006; World Scientific and Engineering Academy and Society (WSEAS): Stevens Point, WI, USA, 2006; pp. 268–275.
- Lloyd, S.P. Least Squares Quantization in PCM. *IEEE Trans. Inf. Theory* **1982**, *28*, 129–137. [\[CrossRef\]](#)
- MacQueen, J.B. Some Methods of Classification and Analysis of Multivariate Observations. In Proceedings of the 5th Berkley Symposium on Mathematical Statistics and Probability, Berkeley, CA, USA, 21 June–18 July 1965 and 27 December 1965–7 January 1966; Volume 1, pp. 281–297.
- Drineas, P.; Frieze, A.; Kannan, R.; Vempala, S.; Vinay, V. Clustering large graphs via the singular value decomposition. *Mach. Learn.* **2004**, *56*, 9–33. [\[CrossRef\]](#)
- Gu, Y.; Li, K.; Guo, Z.; Wang, Y. Semi-supervised k-means ddos detection method using hybrid feature selection algorithm. *IEEE Access* **2019**, *7*, 351–365. [\[CrossRef\]](#)
- Guo, X.; Zhang, X.; He, Y.; Jin, Y.; Qin, H.; Azhar, M.; Huang, J.Z. A Robust k-Means Clustering Algorithm Based on Observation Point Mechanism. *Complexity* **2020**, *2020*, 3650926. [\[CrossRef\]](#)
- Milligan, G.W. Clustering validation: Results and implications for applied analyses. In *Clustering and Classification*; Arabie, P., Hubert, L.J., Soete, G., Eds.; World Scientific: River Edge, NJ, USA, 1996; pp. 341–375.
- Steinley, D.; Brusco, M. Choosing the Number of Clusters in K-Means Clustering. *Psychol. Methods* **2011**, *16*, 285–297. [\[CrossRef\]](#) [\[PubMed\]](#)
- Garey, M.; Johnson, D.; Witsenhausen, H. The complexity of the generalized Lloyd—Max problem (Corresp.) *IEEE Trans. Inf. Theory* **1982**, *28*, 255–256. [\[CrossRef\]](#)
- Aloise, D.; Deshpande, A.; Hansen, P.; Popat, P. NP-hardness of Euclidean sum-of-squares clustering. *Mach. Learn.* **2009**, *75*, 245–248. [\[CrossRef\]](#)
- Cooper, L. Heuristic methods for location-allocation problems. *SIAM Rev.* **1964**, *6*, 37–53. [\[CrossRef\]](#)
- Jiang, J.L.; Yuan, X.M. A heuristic algorithm for constrained multi-source Weber problem. The variational inequality approach. *Eur. J. Oper. Res.* **2007**, *187*, 357–370. [\[CrossRef\]](#)
- Arthur, D.; Manthey, B.; Roglin, H. k-Means Has Polynomial Smoothed Complexity. In Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science (FOCS'09), Atlanta, GA, USA, 25–27 October 2009; IEEE Computer Society: Washington, DC, USA, 2009; pp. 405–414. [\[CrossRef\]](#)
- Sabin, M.J.; Gray, R.M. Global convergence and empirical consistency of the generalized Lloyd algorithm. *IEEE Trans. Inf. Theory* **1986**, *32*, 148–155. [\[CrossRef\]](#)
- Emelianenko, M.; Ju, L.; Rand, A. Nondegeneracy and Weak Global Convergence of the Lloyd Algorithm in Rd. *SIAM J. Numer. Anal.* **2009**, *46*, 1423–1441. [\[CrossRef\]](#)
- Pham, D.T.; Afify, A.A. Clustering techniques and their applications in engineering. Proceedings of the Institution of Mechanical Engineers, Part C. *J. Mech. Eng. Sci.* **2007**, *221*, 1445–1459. [\[CrossRef\]](#)
- Fisher, D.; Xu, L.; Carnes, J.R.; Reich, Y.; Fenves, J.; Chen, J.; Shiavi, R.; Biswas, G.; Weinberg, J. Applying AI clustering to engineering tasks. *IEEE Expert* **1993**, *8*, 51–60. [\[CrossRef\]](#)

20. Gheorghe, G.; Cartina, G.; Rotaru, F. Using K-Means Clustering Method in Determination of the Energy Losses Levels from Electric Distribution Systems. In Proceedings of the International Conference on Mathematical Methods and Computational Techniques in Electrical Engineering, Timisoara, Romania, 21–23 October 2010; pp. 52–56.
21. Kersten, P.R.; Lee, J.S.; Ainsworth, T.L. Unsupervised classification of polarimetric synthetic aperture radar images using fuzzy clustering and EM clustering. *IEEE Trans. Geosci. Remote Sens.* **2005**, *43*, 519–527. [\[CrossRef\]](#)
22. Cesarotti, V.; Rossi, L.; Santoro, R. A neural network clustering model for miscellaneous components production planning. *Prod. Plan. Control* **1999**, *10*, 305–316. [\[CrossRef\]](#)
23. Kundu, B.; White, K.P., Jr.; Mastrangelo, C. Defect clustering and classification for semiconductor devices. In Proceedings of the 45th Midwest Symposium on Circuits and Systems, Tulsa, Oklahoma, 4–7 August 2002; Volume 2, pp. II-561–II-564. [\[CrossRef\]](#)
24. Vernet, A.; Kopp, G.A. Classification of turbulent flow patterns with fuzzy clustering. *Eng. Appl. Artif. Intell.* **2002**, *15*, 315–326. [\[CrossRef\]](#)
25. Afify, A.A.; Dimov, S.; Naim, M.M.; Valeva, V. Detecting cyclical disturbances in supply networks using data mining techniques. In Proceedings of the 2nd European Conference on Management of Technology, Birmingham, UK, 10–12 September 2006; pp. 1–8. [\[CrossRef\]](#)
26. Jain, A.K.; Murty, M.N.; Flynn, P.J. Data clustering: A review. *ACM Comput. Surv.* **1999**, *31*, 264–323. [\[CrossRef\]](#)
27. Naranjo, J.E.; Saha, R.; Tariq, M.T.; Hadi, M.; Xiao, Y. Pattern Recognition Using Clustering Analysis to Support Transportation System Management, Operations, and Modeling. *J. Adv. Transp.* **2019**. [\[CrossRef\]](#)
28. Kadir, R.A.; Shima, Y.; Sulaiman, R.; Ali, F. Clustering of public transport operation using K-means. In Proceedings of the 2018 IEEE 3rd International Conference on Big Data Analysis (ICBDA), Shanghai, China, 9–12 March 2018; pp. 427–532.
29. Sesham, A.; Padmanabham, P.; Govardhan, A. Application of Factor Analysis to k-means Clustering Algorithm on Transportation Data. *IJCA* **2014**, *95*, 40–46. [\[CrossRef\]](#)
30. Deb Nath, R.P.; Lee, H.J.; Chowdhury, N.K.; Chang, J.W. Modified K-Means Clustering for Travel Time Prediction Based on Historical Traffic Data. *LNCS* **2010**, *6276*, 511–521. [\[CrossRef\]](#)
31. Montazeri-Gh, M.; Fotouhi, A. Traffic condition recognition using the k-means clustering method. *Sci. Iran.* **2011**, *18*, 930–937. [\[CrossRef\]](#)
32. Farahani, R.Z.; Hekmatfar, M. *Facility Location Concepts, Models, Algorithms and Case Studies*; Springer: Berlin/Heidelberg, Germany, 2009. [\[CrossRef\]](#)
33. Drezner, Z.; Hamacher, H. *Facility Location: Applications and Theory*; Springer: Berlin, Germany, 2004; pp. 119–143.
34. Klastorin, T.D. The p-Median Problem for Cluster Analysis: A Comparative Test Using the Mixture Model Approach. *Manag. Sci.* **1985**, *31*, 84–95. [\[CrossRef\]](#)
35. Brusco, M.J.; Kohn, H.F. Optimal Partitioning of a Data Set Based on the p-Median Model. *Psychometrika* **2008**, *73*, 89–105. [\[CrossRef\]](#)
36. Kaufman, L.; Rousseeuw, P.J. Clustering by means of Medoids. In *Statistical Data Analysis Based on the L1-Norm and Related Methods*; Dodge, Y., Ed.; Birkhäuser Basel: Basel, Switzerland, 1987; pp. 405–416.
37. Schubert, E.; Rousseeuw, P. Faster k-Medoids Clustering: Improving the PAM, CLARA, and CLARANS Algorithms. *arXiv* **2019**, arXiv:1810.05691.
38. Park, H.S.; Jun, C.H. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl.* **2009**, *36*, 3336–3341. [\[CrossRef\]](#)
39. Hakimi, S.L. Optimum Locations of Switching Centers and the Absolute Centers and Medians of a Graph. *Oper. Res.* **1964**, *12*, 450–459. [\[CrossRef\]](#)
40. Masuyama, S.; Ibaraki, T.; Hasegawa, T. The Computational Complexity of the m-Center Problems on the Plane. *Trans. Inst. Electron. Commun. Eng. Japan* **1981**, *64E*, 57–64.
41. Kariv, O.; Hakimi, S.L. An Algorithmic Approach to Network Location Problems. II: The P medians. *SIAM J. Appl. Math.* **1979**, *37*, 539–560. [\[CrossRef\]](#)
42. Kuenne, R.E.; Soland, R.M. Exact and approximate solutions to the multisource Weber problem. *Math. Program.* **1972**, *3*, 193–209. [\[CrossRef\]](#)
43. Ostresh, L.M., Jr. The Stepwise Location Allocation Problem: Exact Solutions in Continuous and Discrete Spaces. *Geogr. Anal.* **1978**, *10*, 174–185. [\[CrossRef\]](#)

44. Rosing, K.E. An optimal method for solving the (generalized) multi-Weber problem. *Eur. J. Oper. Res.* **1992**, *58*, 414–426. [\[CrossRef\]](#)
45. Blum, C.; Roli, A. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.* **2001**, *35*, 268–308. [\[CrossRef\]](#)
46. Neema, M.N.; Maniruzzaman, K.M.; Ohgai, A. New Genetic Algorithms Based Approaches to Continuous p-Median Problem. *Netw. Spat. Econ.* **2011**, *11*, 83–99. [\[CrossRef\]](#)
47. Hoos, H.H.; Stutzle, T. *Stochastic Local Search Foundations and Applications*; Springer: Berlin, Germany, 2005.
48. Bang-Jensen, J.; Chiarandini, M.; Goegebeur, Y.; Jorgensen, B. Mixed Models for the Analysis of Local Search Components. In Proceedings of the Engineering Stochastic Local Search Algorithms International Workshop, Brussels, Belgium, 6–8 September 2007; pp. 91–105.
49. Cohen-Addad, V.; Mathieu, C. Effectiveness of local search for geometric optimization. In Proceedings of the 31st International Symposium on Computational Geometry, SoCG-2015, Eindhoven, The Netherlands, 22–25 June 2015; pp. 329–343.
50. Kochetov, Y.; Mladenović, N.; Hansen, P. Local search with alternating neighborhoods. *Discret. Anal. Oper. Res.* **2003**, *2*, 11–43. (In Russian)
51. Kanungo, T.; Mount, D.M.; Netanyahu, N.S.; Piatko, C.D.; Silverman, R.; Wu, A.Y. A local search approximation algorithm for k-means clustering. *Comput. Geom. Theory Appl.* **2004**, *28*, 89–112. [\[CrossRef\]](#)
52. Page, E.S. On Monte Carlo methods in congestion problems. I: Searching for an optimum in discrete situations. *Oper. Res.* **1965**, *13*, 291–299. [\[CrossRef\]](#)
53. Hromkovic, J. *Algorithmics for Hard Problems: Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics*; Springer: Berlin/Heidelberg, Germany, 2011.
54. Ng, T. Expanding Neighborhood Tabu Search for facility location problems in water infrastructure planning. In Proceedings of the 2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC), San Diego, CA, USA, 5–8 October 2014; pp. 3851–3854. [\[CrossRef\]](#)
55. Mladenovic, N.; Brimberg, J.; Hansen, P.; Moreno-Perez, J.A. The p-median problem: A survey of metaheuristic approaches. *Eur. J. Oper. Res.* **2007**, *179*, 927–939. [\[CrossRef\]](#)
56. Reese, J. Solution methods for the p-median problem: An annotated bibliography. *Networks* **2006**, *48*, 125–142. [\[CrossRef\]](#)
57. Brimberg, J.; Drezner, Z.; Mladenovic, N.; Salhi, S. A New Local Search for Continuous Location Problems. *Eur. J. Oper. Res.* **2014**, *232*, 256–265. [\[CrossRef\]](#)
58. Drezner, Z.; Brimberg, J.; Mladenovic, N.; Salhi, S. New heuristic algorithms for solving the planar p-median problem. *Comput. Oper. Res.* **2015**, *62*, 296–304. [\[CrossRef\]](#)
59. Drezner, Z.; Brimberg, J.; Mladenovic, N.; Salhi, S. Solving the planar p-median problem by variable neighborhood and concentric searches. *J. Glob. Optim.* **2015**, *63*, 501–514. [\[CrossRef\]](#)
60. Arthur, D.; Vassilvitskii, S. k-Means++: The Advantages of Careful Seeding. In Proceedings of the SODA'07, SIAM, New Orleans, LA, USA, 7–9 January 2007; pp. 1027–1035.
61. Bradley, P.S.; Fayyad, U.M. Refining Initial Points for K-Means Clustering. In Proceedings of the Fifteenth International Conference on Machine Learning (ICML '98), Madison, WI, USA, 24–27 July 1998; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1998; pp. 91–99.
62. Bhusare, B.B.; Bansode, S.M. Centroids Initialization for K-Means Clustering using Improved Pillar Algorithm. *Int. J. Adv. Res. Comput. Eng. Technol.* **2014**, *3*, 1317–1322.
63. Yang, J.; Wang, J. Tag clustering algorithm Immsk: Improved k-means algorithm based on latent semantic analysis. *J. Syst. Electron.* **2017**, *28*, 374–384.
64. Mishra, N.; Oblinger, D.; Pitt, L. Sublinear time approximate clustering. In Proceedings of the 12th SODA, Washington, DC, USA, 7–9 January 2001; pp. 439–447.
65. Eisenbrand, F.; Grandoni, F.; Rothvoss, T.; Schafer, G. Approximating connected facility location problems via random facility sampling and core detouring. In Proceedings of the SODA'2008, San Francisco, CA, USA, 20–22 January 2008; ACM: New York, NY, USA, 2008; pp. 1174–1183. [\[CrossRef\]](#)
66. Jaiswal, R.A.; Kumar, A.; Sen, S. Simple D2-Sampling Based PTAS for k-Means and Other Clustering Problems. *Algorithmica* **2014**, *70*, 22–46. [\[CrossRef\]](#)
67. Avella, P.; Boccia, M.; Salerno, S.; Vasilyev, I. An Aggregation Heuristic for Large Scale p-median Problem. *Comput. Oper. Res.* **2012**, *39*, 1625–1632. [\[CrossRef\]](#)

68. Kaufman, L.; Rousseeuw, P.J. *Finding Groups in Data: An Introduction to Cluster Analysis*; Wiley: New York, NY, USA, 1990.
69. Francis, R.L.; Lowe, T.J.; Rayco, M.B.; Tamir, A. Aggregation error for location models: Survey and analysis. *Ann. Oper. Res.* **2009**, *167*, 171–208. [\[CrossRef\]](#)
70. Pelleg, D.; Moore, A. *Accelerating Exact k-Means with Geometric Reasoning* [Technical Report CMU-CS-00-105]; Carnegie Mellon University: Pittsburgh, PA, USA, 2000.
71. Borgelt, C. Even Faster Exact k-Means Clustering. *LNCS* **2020**, *12080*, 93–105. [\[CrossRef\]](#)
72. Lai, J.Z.C.; Huang, T.J.; Liaw, Y.C. A Fast k-Means Clustering Algorithm Using Cluster Center Displacement. *Pattern Recognit.* **2009**, *42*, 2551–2556. [\[CrossRef\]](#)
73. Mladenovic, N.; Hansen, P. Variable Neighborhood Search. *Comput. Oper. Res.* **1997**, *24*, 1097–1100. [\[CrossRef\]](#)
74. Hansen, P. Variable Neighborhood Search. Search Methodology. In *Search Methodologies*; Bruke, E.K., Kendall, G., Eds.; Springer: New York, NY, USA, 2005; pp. 211–238. [\[CrossRef\]](#)
75. Hansen, P.; Mladenovic, N. Variable Neighborhood Search. In *Handbook of Heuristics*; Martí, R., Pardalos, P., Resende, M., Eds.; Springer: Cham, Switzerland, 2018. [\[CrossRef\]](#)
76. Brimberg, J.; Hansen, P.; Mladenovic, N. Attraction Probabilities in Variable Neighborhood Search. *4OR-Q. J. Oper. Res.* **2010**, *8*, 181–194. [\[CrossRef\]](#)
77. Hansen, P.; Mladenovic, N.; Perez, J.A.M. Variable Neighborhood Search: Methods and Applications. *4OR-Q. J. Oper. Res.* **2008**, *6*, 319–360. [\[CrossRef\]](#)
78. Hansen, P.; Brimberg, J.; Urosevic, D.; Mladenovic, N. Solving Large p-Median Clustering Problems by Primal Dual Variable Neighborhood Search. *Data Min. Knowl. Discov.* **2009**, *19*, 351–375. [\[CrossRef\]](#)
79. Rozhnov, I.P.; Orlov, V.I.; Kazakovtsev, L.A. VNS-Based Algorithms for the Centroid-Based Clustering Problem. *Facta Univ. Ser. Math. Inform.* **2019**, *34*, 957–972.
80. Hansen, P.; Mladenovic, N. J-Means: A new local search heuristic for minimum sum-of-squares clustering. *Pattern Recognit.* **2001**, *34*, 405–413. [\[CrossRef\]](#)
81. Martins, P. Goal Clustering: VNS Based Heuristics. Available online: <https://arxiv.org/abs/1705.07666v4> (accessed on 24 October 2020).
82. Carrizosa, E.; Mladenovic, N.; Todosijevic, R. Variable neighborhood search for minimum sum-of-squares clustering on networks. *Eur. J. Oper. Res.* **2013**, *230*, 356–363. [\[CrossRef\]](#)
83. Roux, M. A Comparative Study of Divisive and Agglomerative Hierarchical Clustering Algorithms. *J. Classif.* **2018**, *35*, 345–366. [\[CrossRef\]](#)
84. Sharma, A.; López, Y.; Tsunoda, T. Divisive hierarchical maximum likelihood clustering. *BMC Bioinform.* **2017**, *18*, 546. [\[CrossRef\]](#)
85. Venkat Reddy, M.; Vivekananda, M.; Satish, R.U.V.N. Divisive Hierarchical Clustering with K-means and Agglomerative Hierarchical Clustering. *IJCST* **2017**, *5*, 6–11.
86. Sun, Z.; Fox, G.; Gu, W.; Li, Z. A parallel clustering method combined information bottleneck theory and centroid-based clustering. *J. Supercomput.* **2014**, *69*, 452–467. [\[CrossRef\]](#)
87. Kuehn, A.A.; Hamburger, M.J. A heuristic program for locating warehouses. *Manag. Sci.* **1963**, *9*, 643–666. [\[CrossRef\]](#)
88. Alp, O.; Erkut, E.; Drezner, Z. An Efficient Genetic Algorithm for the p-Median Problem. *Ann. Oper. Res.* **2003**, *122*, 21–42. [\[CrossRef\]](#)
89. Cheng, J.; Chen, X.; Yang, H.; Leng, M. An enhanced k-means algorithm using agglomerative hierarchical clustering strategy. In Proceedings of the International Conference on Automatic Control and Artificial Intelligence (ACAI 2012), Xiamen, China, 3–5 March 2012; pp. 407–410. [\[CrossRef\]](#)
90. Kazakovtsev, L.A.; Antamoshkin, A.N. Genetic Algorithm with Fast Greedy Heuristic for Clustering and Location Problems. *Informatica* **2014**, *3*, 229–240.
91. Pelleg, D.; Moore, A. X-means: Extending K-means with Efficient Estimation of the Number of Clusters. In Proceedings of the International Conference on Machine Learning ICML, Sydney, Australia, 8–12 July 2002.
92. Ahmed, M.; Seraj, R.; Islam, S.M.S. The k-means Algorithm: A Comprehensive Survey and Performance Evaluation. *Electronics* **2020**, *9*, 1295. [\[CrossRef\]](#)
93. Frackiewicz, M.; Mandrella, A.; Palus, H. Fast Color Quantization by K-Means Clustering Combined with Image Sampling. *Symmetry* **2019**, *11*, 963. [\[CrossRef\]](#)
94. Zhang, G.; Li, Y.; Deng, X. K-Means Clustering-Based Electrical Equipment Identification for Smart Building Application. *Information* **2020**, *11*, 27. [\[CrossRef\]](#)

95. Chen, F.; Yang, Y.; Xu, L.; Zhang, T.; Zhang, Y. Big-Data Clustering: K-Means or K-Indicators? 2019. Available online: <https://arxiv.org/pdf/1906.00938.pdf> (accessed on 18 October 2020).
96. Qin, J.; Fu, W.; Gao, H.; Zheng, W.X. Distributed k-means algorithm and fuzzy c-means algorithm for sensor networks based on multiagent consensus theory. *IEEE Trans. Cybern.* **2016**, *47*, 772–783. [CrossRef]
97. Shindler, M.; Wong, A.; Meyerson, A. Fast and accurate k-means for large datasets. In Proceedings of the 24th International Conference on Neural Information Processing Systems (NIPS'11), Sydney, Australia, 13–16 December 2011; Curran Associates Inc.: Red Hook, NY, USA, 2011; pp. 2375–2383.
98. Hedar, A.R.; Ibrahim, A.M.M.; Abdel-Hakim, A.E.; Sewisy, A.A. K-Means Cloning: Adaptive Spherical K-Means Clustering. *Algorithms* **2018**, *11*, 151. [CrossRef]
99. Xu, T.S.; Chiang, H.D.; Liu, G.Y.; Tan, C.W. Hierarchical k-means method for clustering large-scale advanced metering infrastructure data. *IEEE Trans. Power Deliv.* **2015**, *32*, 609–616. [CrossRef]
100. Wang, X.D.; Chen, R.C.; Yan, F.; Zeng, Z.Q.; Hong, C.Q. Fast adaptive k-means subspace clustering for high-dimensional data. *IEEE Access* **2019**, *7*, 639–651. [CrossRef]
101. Zechner, M.; Granitzer, M. Accelerating K-Means on the Graphics Processor via CUDA. In Proceedings of the International Conference on Intensive Applications and Services, Valencia, Spain, 20–25 April 2009; pp. 7–15. [CrossRef]
102. Luebke, D.; Humphreys, G. How GPUs work. *Computer* **2007**, *40*, 96–110. [CrossRef]
103. Maulik, U.; Bandyopadhyay, S. Genetic Algorithm-Based Clustering Technique. *Pattern Recognit.* **2000**, *33*, 1455–1465. [CrossRef]
104. Krishna, K.; Murty, M. Genetic K-Means algorithm. *IEEE Trans. Syst. Man Cybern. Part B* **1999**, *29*, 433–439. [CrossRef] [PubMed]
105. Singh, N.; Singh, D.P.; Pant, B. ACOCA: Ant Colony Optimization Based Clustering Algorithm for Big Data Preprocessing. *Int. J. Math. Eng. Manag. Sci.* **2019**, *4*, 1239–1250. [CrossRef]
106. Merwe, D.W.; Engelbrecht, A.P. Data Clustering Using Particle Swarm Optimization. In Proceedings of the 2003 Congress on Evolutionary Computation, Canberra, Australia, 8–12 December 2003; pp. 215–220.
107. Nikolaev, A.; Mladenovic, N.; Todosijevic, R. J-means and I-means for minimum sum-of-squares clustering on networks. *Optim. Lett.* **2017**, *11*, 359–376. [CrossRef]
108. Fränti, P.; Sieranoja, S. K-means properties on six clustering benchmark datasets. *Appl. Intell.* **2018**, *48*, 4743–4759. [CrossRef]
109. Clustering Basic Benchmark. Available online: <http://cs.joensuu.fi/sipu/datasets/> (accessed on 15 September 2020).
110. Kazakovtsev, L.; Shkaberina, G.; Rozhnov, I.; Li, R.; Kazakovtsev, V. Genetic Algorithms with the Crossover-Like Mutation Operator for the k-Means Problem. *CCIS* **2020**, *1275*, 350–362. [CrossRef]
111. Brimberg, J.; Mladenovic, N. A variable neighborhood algorithm for solving the continuous location-allocation problem. *Stud. Locat. Anal.* **1996**, *10*, 1–12.
112. Miskovic, S.; Stanimirovich, Z.; Grujicic, I. An efficient variable neighborhood search for solving a robust dynamic facility location problem in emergency service network. *Electron. Notes Discret. Math.* **2015**, *47*, 261–268. [CrossRef]
113. Crainic, T.G.; Gendreau, M.; Hansen, P.; Hoeb, N.; Mladenovic, N. Parallel variable neighbourhood search for the p-median. In Proceedings of the 4th Metaheuristics International conference MIC'2001, Porto, Portugal, 16–21 July 2001; pp. 595–599.
114. Hansen, P.; Mladenovic, N. Variable neighborhood search for the p-median. *Locat. Sci.* **1997**, *5*, 207–226. [CrossRef]
115. Wen, M.; Krappner, E.; Larsen, J.; Stidsen, T.K. A multilevel variable neighborhood search heuristic for a practical vehicle routing and driver scheduling problem. *Networks* **2011**, *58*, 311–323. [CrossRef]
116. Baldassi, C. Recombinator-k-Means: Enhancing k-Means++ by Seeding from Pools of Previous Runs. Available online: <https://arxiv.org/abs/1905.00531v1> (accessed on 18 September 2020).
117. Duarte, A.; Mladenović, N.; Sánchez-Oro, J.; Todosijević, R. Variable Neighborhood Descent. In *Handbook of Heuristics*; Martí, R., Panos, P., Resende, M., Eds.; Springer: Cham, Switzerland, 2016. [CrossRef]
118. Dua, D.; Graff, C. UCI Machine Learning Repository 2019. Available online: <http://archive.ics.uci.edu/ml> (accessed on 30 September 2020).
119. Molla, M.M.; Nag, P.; Thohura, S.; Khan, A. A Graphics Process Unit-Based Multiple-Relaxation-Time Lattice Boltzmann Simulation of Non-Newtonian Fluid Flows in a Backward Facing Step. *Computation* **2020**, *8*, 83. [CrossRef]

120. Kazakovtsev, L.A.; Rozhnov, I.P.; Popov, E.A.; Karaseva, M.V.; Stupina, A.A. Parallel implementation of the greedy heuristic clustering algorithms. *IOP Conf. Ser. Mater. Sci. Eng.* **2019**, *537*, 022052. [[CrossRef](#)]
121. Zhang, T.; Ramakrishnan, R.; Livny, M. BIRCH: An Efficient Data Clustering Method for Very Large Databases. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of data (SIGMOD'96), Montreal, QC, Canada, 4–6 June 1996; ACM: New York, NY, USA, 1996; pp. 103–114. [[CrossRef](#)]
122. Smucker, M.D.; Allan, J.; Carterette, B.A. Comparison of Statistical Significance Tests for Information Retrieval. In Proceedings of the Sixteenth ACM Conference on Information and Knowledge Management (CIKM '07), Lisbon, Portugal, 6–10 November 2007; ACM: New York, NY, USA, 2007; pp. 623–632.
123. Park, H.M. *Comparing Group Means: The t-Test and One-way ANOVA Using STATA, SAS, and SPSS*; Indiana University: Bloomington, Indiana, 2009.
124. Mann, H.B.; Whitney, D.R. On a Test of Whether one of Two Random Variables is Stochastically Larger than the other. *Ann. Math. Stat.* **1947**, *18*, 50–60. [[CrossRef](#)]
125. Fay, M.P.; Proschan, M.A. Wilcoxon-Mann-Whitney or t-Test? On Assumptions for Hypothesis Tests and Multiple Interpretations of Decision Rules. *Stat. Surv.* **2010**, *4*, 1–39. [[CrossRef](#)]
126. Burke, E.; Gendreau, M.; Hyde, M.; Kendall, G.; Ochoa, G.; Ozkan, E.; Qu, R. Hyper-heuristics: A survey of the state of the art. *J. Oper. Res. Soc.* **2013**, *64*, 1695–1724. [[CrossRef](#)]
127. Stanovov, V.; Semekin, E.; Semenkina, O. Self-configuring hybrid evolutionary algorithm for fuzzy imbalanced classification with adaptive instance selection. *J. Artif. Intell. Soft Comput. Res.* **2016**, *6*, 173–188. [[CrossRef](#)]
128. Semenkina, M.; Semekin, E. Hybrid Self-configuring Evolutionary Algorithm for Automated Design of Fuzzy Classifier. *LNCS* **2014**, *8794*, 310–317. [[CrossRef](#)]

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).